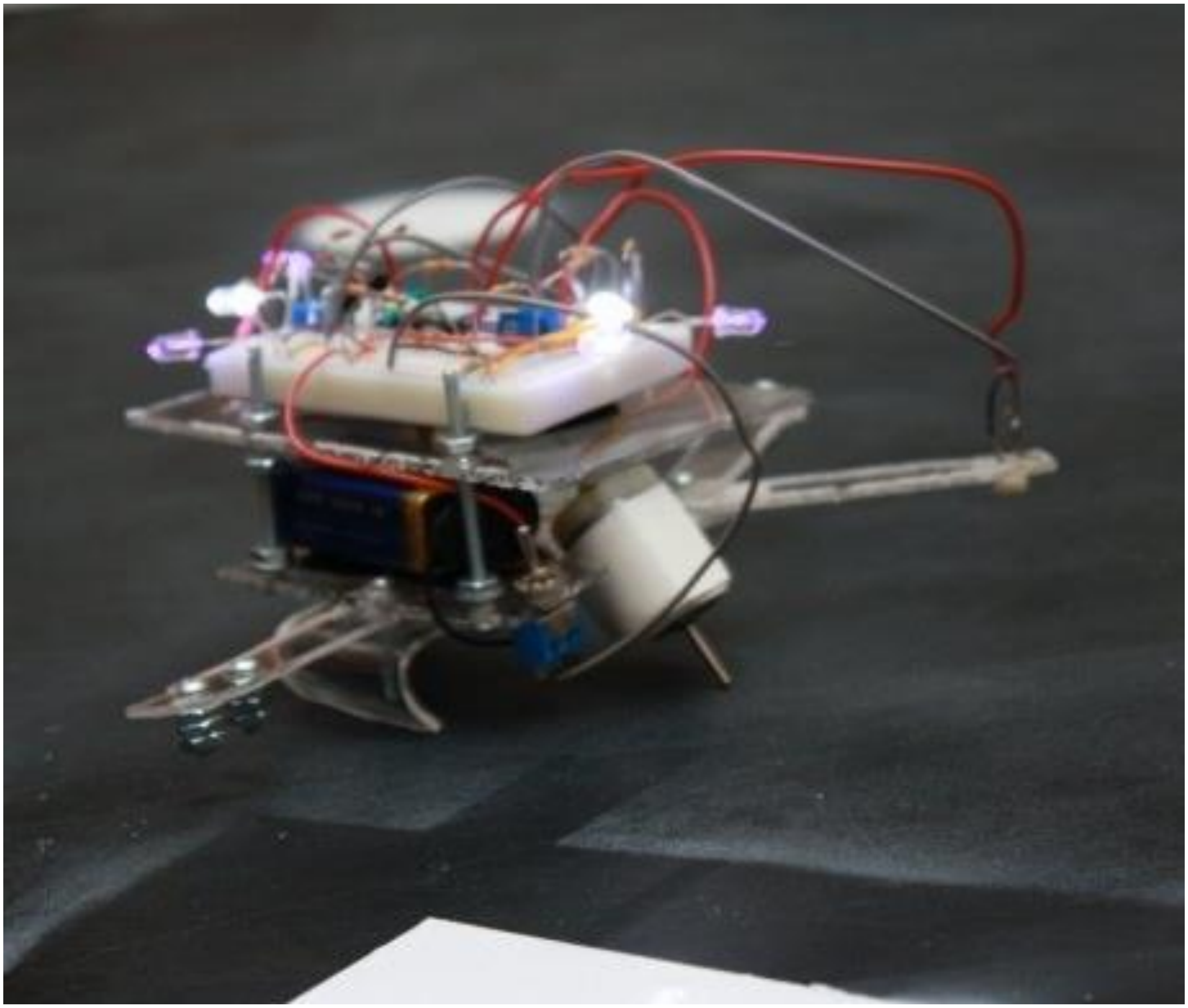


Electromechanical Systems

1st Edition

Chad Davis, PhD, PE

12/12/18

Electromechanical Systems, 1st Edition.

How to navigate this book: This document contains **bookmark** links, which are links to other locations in this document. These are not the same as **website links**, which will open up a webpage in your default browser. If you click on a bookmark and want to go back to where you were previously at in the document, click on **ALT - Left Arrow (←)**.

For example:

- This is a **bookmark** to the Preface Section → Preface Appendix – Dependent
- This is a **website link** to the latest version of this book → EMS Book

Getting the Latest Version of this book: If you see any errors or typos please go to my website (ChadDavis.oucreate.com) and enter them in the comments section and I will get the book updated. Minor changes in the book will not result in a new version number, but instead will be corrected and uploaded to the following webpage: https://shareok.org/handle/11244/316816

If you have any suggestions for significant changes that could potentially be implemented into a new edition in the future, I would also appreciate that type of feedback on my website.

Table of Contents

Preface

This eBook was written as the third installment in the series that coincide with three engineering courses taught at the University of Oklahoma (ENGR 2431, ENGR 2531, and ENGR 3431). These courses were designed to provide non-major students – those not majoring in electrical or computer engineering (ECE) – a foundation in various ECE topics. ENGR 2431 is a prerequisite for both ENGR 2531 and ENGR 3431 and it is recommended that the DC Circuits book be studied prior to beginning the eBooks created for the other two courses.

The first eBook, titled ***DC Circuits***, was written in 2016 to serve as the textbook for the course titled *ENGR 2431 – DC Circuits*. Some of the main topics covered in the DC Circuits book are:

- Introductory concepts in DC circuits, such as resistance, charge, voltage, current, and power.
- Basic Theorems – Ohm's Law, Kirchhoff's Voltage Law, and Kirchhoff's Current Law.
- Combining resistors that are in series and parallel.
- Advanced Problem Solving Techniques – Mesh and Nodal Analysis (with an emphasis on using matrices).
- Source transformations and superposition analysis of DC circuits.
- Thevenin and Norton equivalent circuits.
- Introduction to capacitors and inductors.
- Transient and steady state analysis of RLC circuits with DC sources.

The second eBook, titled ***AC Circuits***, was written in 2017 to serve as the textbook for the course titled *ENGR 2531 – AC Circuits*. The AC Circuits course (and book) also includes a brief introduction to electronics. Some of the main topics covered in the AC Circuits book are:

- AC signal overview that includes sinusoids, as well as square waves and triangular waves.
- Review in vector mathematics that is used for AC circuits.
- RLC circuit analysis with AC sources – Series circuits, parallel circuits, and AC power calculations.
- Passive Filters – Low Pass, High Pass, Band Pass, and Band Stop filters created with RLC circuits.
- Basic principles behind transformers and the importance of their use in power systems.
- Introduction to Electronics – Focused primarily on diodes and operational amplifiers.

Multisim was used heavily in the first two books so the circuits that were being analyzed could also be simulated and electrical measuring devices such as Multimeters and Oscilloscopes could be introduced without having a laboratory component to the book (or courses). To allow the non-major students to get hands-on experience in building circuits, a kit is checked out to students in ENGR 2431 that includes a breadboard, Multimeter, and a variety of components. Using this kit, the students are allowed to design, build, simulate, and analyze DC circuits in a semester long project. In the *ENGR 3431 – Electromechanical Systems*, LabVIEW is introduced to help explain many of the various topics and is used for the hands-on project, where students apply the knowledge gained through the semester in digital logic, sensors, and motors to build a robot. The robot is programmed using LabVIEW state machine architecture, describe in the LabVIEW overview.

This eBook, which coincides with the *ENGR 3431 – Electromechanical Systems* course, has a variety of different topics, and when coupled with the topics in the previous two eBooks, a solid ECE knowledge-base is provided. Studying these three eBooks gives the students the background needed to dive into electromechanical system projects. Any student, regardless of major, interested in robotics or mechatronics would also be benefited by studying the topics contained in this eBook trilogy. I would like to thank all of my students who helped edit and improve these books and I am also truly grateful for the sacrifice and support of my wife, Jennifer, and my children, Mitchell and Lindy, during this time-intensive process. Lastly, I would like to thank my Lord and Savior, Jesus Christ for affording me the strength and perseverance to complete this challenging task. [Isaiah 40:29-31]

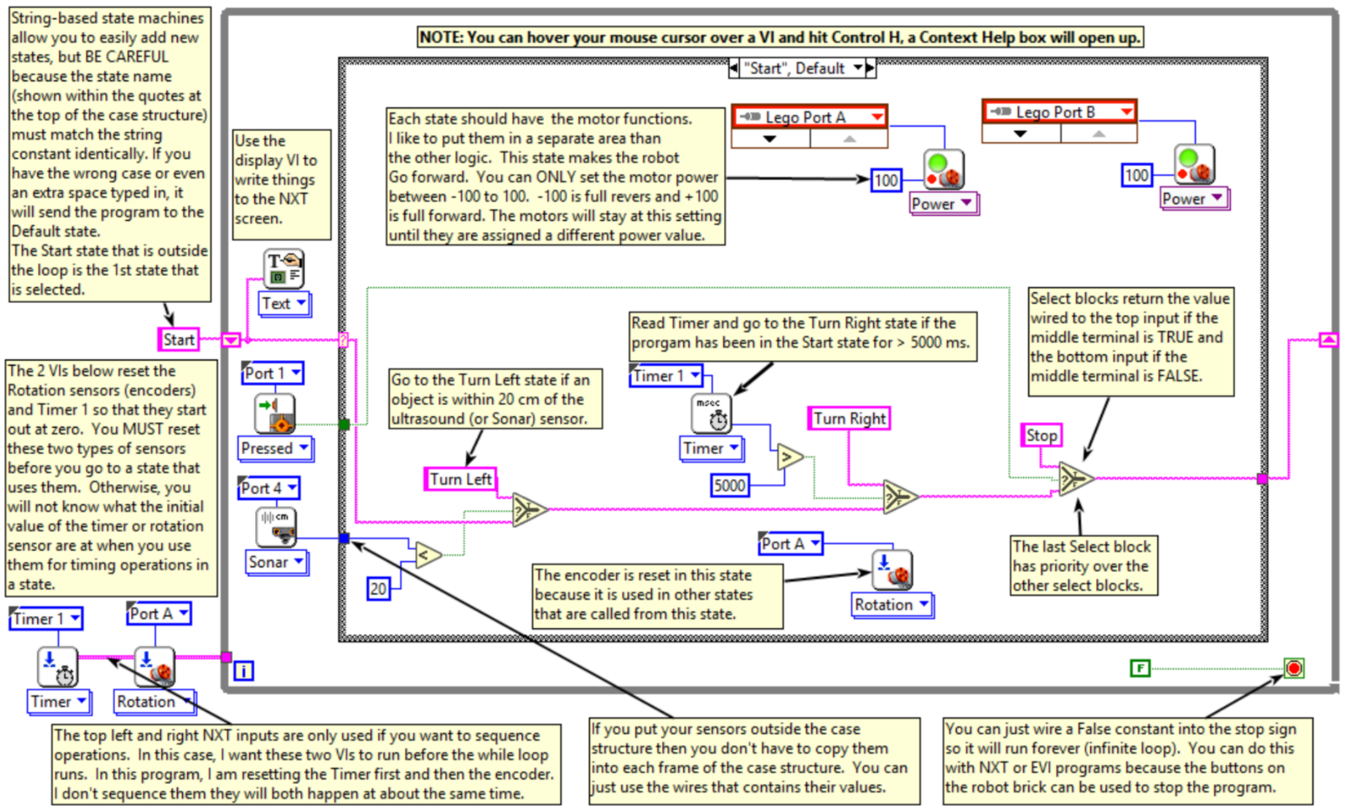
LabVIEW Overview

Before reading this book, an understanding of LabVIEW is desired so the examples and figures in this book are easier to understand. There is also a robot project in ENGR 3431 that requires LabVIEW programming. LabVIEW examples are shown in Figures 0.1, 0.2, 0.3, and 0.4. In my experience, the best way to learn LabVIEW is by diving in and creating a project and using tutorials and online resources to help you figure out how to do the things that are needed in order to complete the project. Some tutorials that will help you get started are shown below. For a more complete list of help documents, google "Learn LabVIEW" and the link shown below should come up that will guide you through all the basics of LabVIEW and even allows you to complete exercises and take an exam to test your abilities and knowledge. http://www.ni.com/academic/students/learn-labview/

The following links have some good tutorials and exercises that go further into different aspects of LabVIEW.

- Loops: http://www.ni.com/white-paper/7588/en/
- Timing, Shift Registers, and Case Structures: http://www.ni.com/white-paper/7592/en/
- Arrays, Clusters, and Text Based Nodes: http://www.ni.com/white-paper/7571/en/
  http://www.ni.com/white-paper/7552/en/
- State Machines: http://www.ni.com/tutorial/7595/en/

The state machine tutorial in the previous bullet contains a vending machine exercise that is recommended to help understand how a LabVIEW state machine architecture works. A LabVIEW program that uses state machine architecture to perform basic functions on a Lego NXT robot is shown in Figures 0.1, 0.2, 0.3, and 0.4. Enumeration constants are used in the vending machine exercise, but string constants are used in the VI below.



String-based state machines allow you to easily add new states, but BE CAREFUL because the state name (shown within the quotes at the top of the case structure) must match the string constant identically. If you have the wrong case or even an extra space typed in, it will send the program to the Default state. The Start state that is outside the loop is the 1st state that is selected.

NOTE: You can hover your mouse cursor over a VI and hit Control H, a Context Help box will open up.

Use the display VI to write things to the NXT screen.

Each state should have the motor functions. I like to put them in a separate area than the other logic. This state makes the robot Go forward. You can ONLY set the motor power between -100 to 100. -100 is full revers and +100 is full forward. The motors will stay at this setting until they are assigned a different power value.

Read Timer and go to the Turn Right state if the prorgam has been in the Start state for > 5000 ms.

Select blocks return the value wired to the top input if the middle terminal is TRUE and the bottom input if the middle terminal is FALSE.

Go to the Turn Left state if an object is within 20 cm of the ultrasound (or Sonar) sensor.

The 2 VIs below reset the Rotation sensors (encoders) and Timer 1 so that they start out at zero. You MUST reset these two types of sensors before you go to a state that uses them. Otherwise, you will not know what the initial value of the timer or rotation sensor are at when you use them for timing operations in a state.

The encoder is reset in this state because it is used in other states that are called from this state.

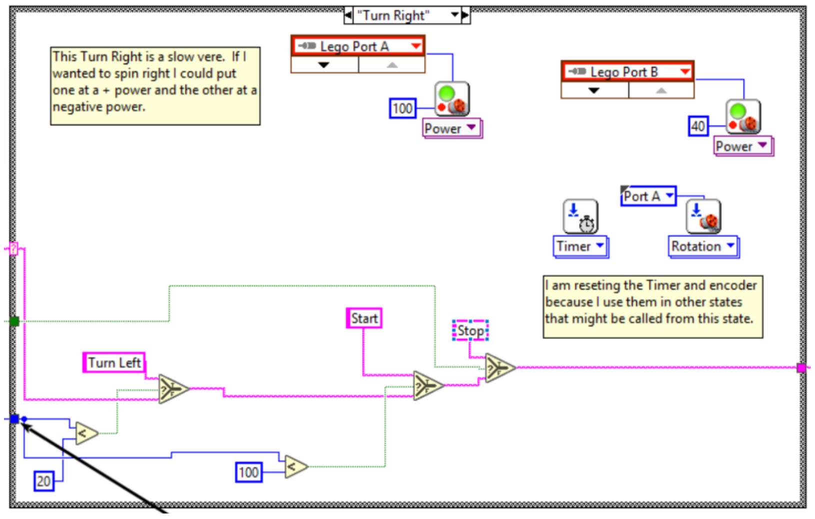
The last Select block has priority over the other select blocks.

The top left and right NXT inputs are only used if you want to sequence operations. In this case, I want these two VIs to run before the while loop runs. In this program, I am resetting the Timer first and then the encoder. I don't sequence them they will both happen at about the same time.

If you put your sensors outside the case structure then you don't have to copy them into each frame of the case structure. You can just use the wires that contains their values.

You can just wire a False constant into the stop sign so it will run forever (infinite loop). You can do this with NXT or EVI programs because the buttons on the robot brick can be used to stop the program.

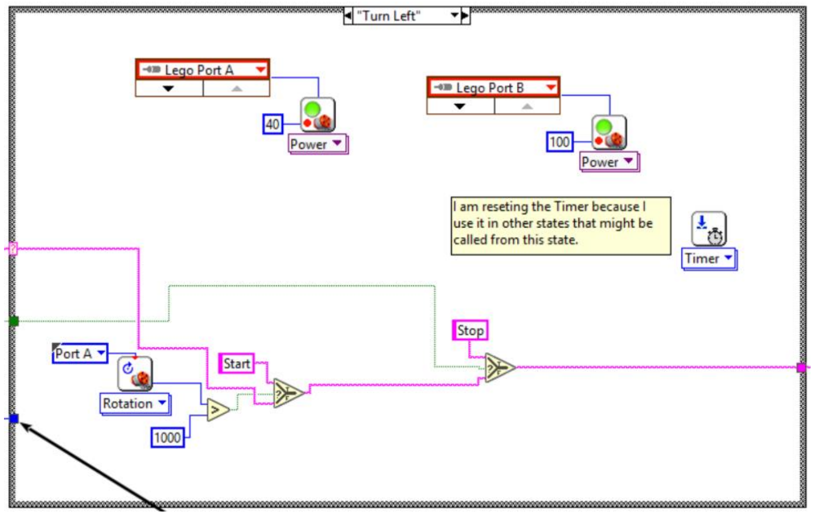
Figure 0.1: LabVIEW NXT State Machine program showing the state machine structure and the "Start" state.



This Turn Right is a slow vere. If I wanted to spin right I could put one at a + power and the other at a negative power.

I am reseting the Timer and encoder because I use them in other states that might be called from this state.

Figure 0.2: "Turn Right" state in the LabVIEW NXT State Machine program from Figure 0.1.



I am reseting the Timer because I use it in other states that might be called from this state.

Figure 0.3: "Turn Left" state in the LabVIEW NXT State Machine program from Figure 0.1.

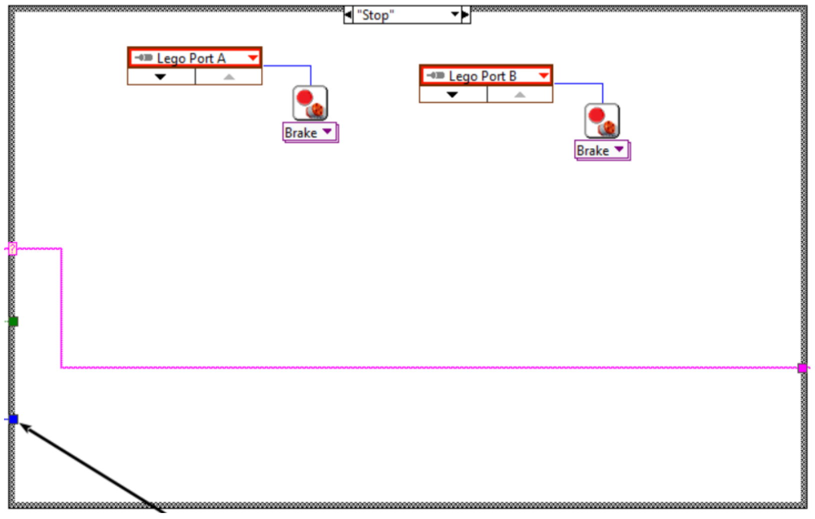


Figure 0.4: "Stop" state in the LabVIEW NXT State Machine program from Figure 0.1.

Table of Contents

Module 1 – Number Systems and Character Encoding

Before discussing digital systems in Module 2, knowledge about different types of number systems and character coding schemes used by computers is necessary. It is commonly known that computers operate by processing information made up of only ones and zeros. Each one or zero is referred to as a **bit**. A "bit" is a combination of the words "**b**inary dig**it**." 8 bits that are grouped together is called a **byte**. Representing a number in terms of a series of ones and zeros is called the binary (or Base 2) number system. It is called "Base 2" because there are only two symbols used (0 and 1). The following example shows a number that is represented in the binary number system and uses 8 bits (or 1 byte) to provide all of the information about that number.

Binary (Base 2) Number Example: $(10101000)_2$ ← the subscript denotes a base 2 number

While computers process information using the binary number system, humans typically prefer to use alternatives. The most popular number system for humans is Decimal (or Base 10) and it was likely created because we usually have 10 fingers and tend to use them early on in our lives to count things. The following is the decimal representation of the Binary Number Example shown above. When binary is used as a representation of a decimal number, it is often called a Binary Coded Decimal (BCD).

Decimal (Base 10) Number Example: $(168)_{10}$ ← the subscript denotes a base 10 number

In order to convert from binary to decimal, the "weight" of each bit must be established. The least significant bit (LSB) of a binary number is the far right bit and it has a weight of 2^0. This means that if the LSB has a value of 1 in it then a value of $1\cdot2^0$ will be added when determining the decimal number that is associated with the number represented in the binary format. The next bit to the left (or 2nd to the right end) has a weight of 2^1, and this pattern of increasing the exponent by one each time you move to the next bit to the left continues for all of the bits that make up the number. The decimal number above can be computed by multiplying each weight (2^0 to 2^7) with the value of the bit (0 or 1) and taking the sum of the results as shown below:

$$1\cdot2^7 + 0\cdot2^6 + 1\cdot2^5 + 0\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 0\cdot2^0 \quad = \quad 1\cdot128 + 0\cdot64 + 1\cdot32 + 0\cdot16 + 1\cdot8 + 0\cdot4 + 0\cdot2 + 0\cdot1 = \mathbf{168}$$

↑ MSB **↑ LSB** **The binary number is shown in blue font (1010 1000)**

In the example above the LSB and the Most Significant Bit (MSB) are labeled. When converting from binary to decimal, the MSB has a weight equal to $\mathbf{2^{N-1}}$ (where N is the number of bits). For example, if there are 16 bits in a binary number the weight of the MSB (far left bit) is 2^{15} and if the MSB has a value of 1 then a value of 2^{15} (32768) is included as one of the 8 terms that sum together to make up the decimal number.

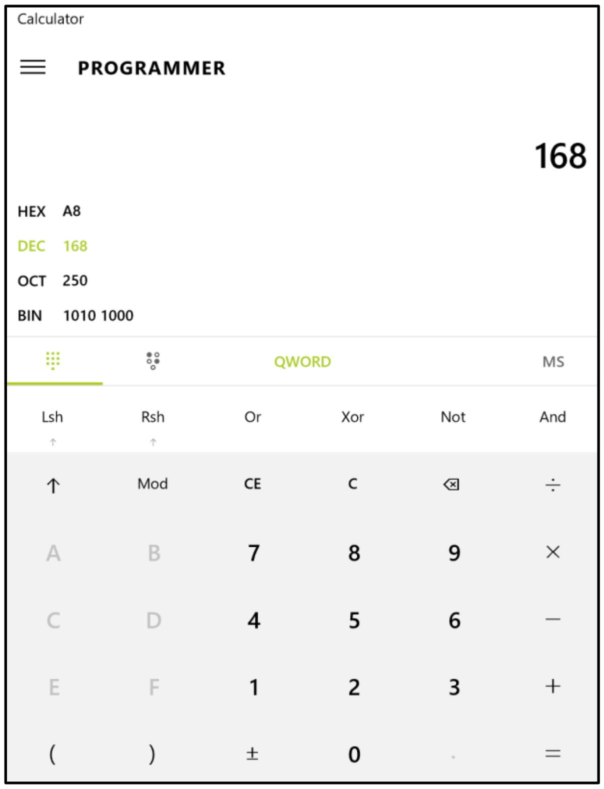
To convert from decimal to binary, a more complicated process is needed. One method that can be used is shown below. The resultant bit values are shown in red font with the MSB coming from the quotient in step 1).

1) 168/128 → Quotient = 1, Remainder = 40 →Divide the decimal number (168) by the weight of the MSB (128)
2) 40/64 → Quotient = 0, Remainder = 40 →Divide the previous remainder (40) by the next bit's weight (64)
3) 40/32 → Quotient = 1, Remainder = 8 →Divide the previous remainder (40) by the next bit's weight (32)
4) 8/16 → Quotient = 0, Remainder = 8 →Divide the previous remainder (8) by the next bit's weight (16)
5) 8/8 → Quotient = 1, Remainder = 0 →Divide the previous remainder (8) by the next bit's weight (8)
6) 0/4 → Quotient = 0, Remainder = 0 →Divide the previous remainder (0) by the next bit's weight (4)
7) 0/2 → Quotient = 0, Remainder = 0 →Divide the previous remainder (0) by the next bit's weight (2)
8) 0/1 → Quotient = 0, Remainder = 0 →Divide the previous remainder (0) by the next bit's weight (1)

- **Final Result: $(168)_{10}$ → $(1010\ 1000)_2$**

It is always a good idea to check your math when doing a multi-step problem like this and fortunately there are many ways to do this. First of all, many calculators have the ability to do this conversion. Figure 1.1 shows a screenshot of a Windows calculator in programming mode. It shows that the decimal value of $(168)_{10}$ is equal to a binary value of $(1010\ 1000)_2$ as calculated in the previous example. LabVIEW is another tool that can be used to convert numbers to different formats. Figure 1.2 also shows that $(168)_{10}$ is equal to $(1010\ 1000)_2$.

Note: In decimal (**Base 10**), there is a place value for ones, tens, hundreds, thousands, etc. This comes from ($\mathbf{10}^0$, $\mathbf{10}^1$, $\mathbf{10}^2$, $\mathbf{10}^3$, etc.). The base stays the same, but the exponent increases by one as you move to the next digit to the left. The same concept applies in binary (Base 2) with ones, twos, fours, eights, etc. (2^0, 2^1, 2^2, 2^3, etc.).


Calculator

PROGRAMMER

168

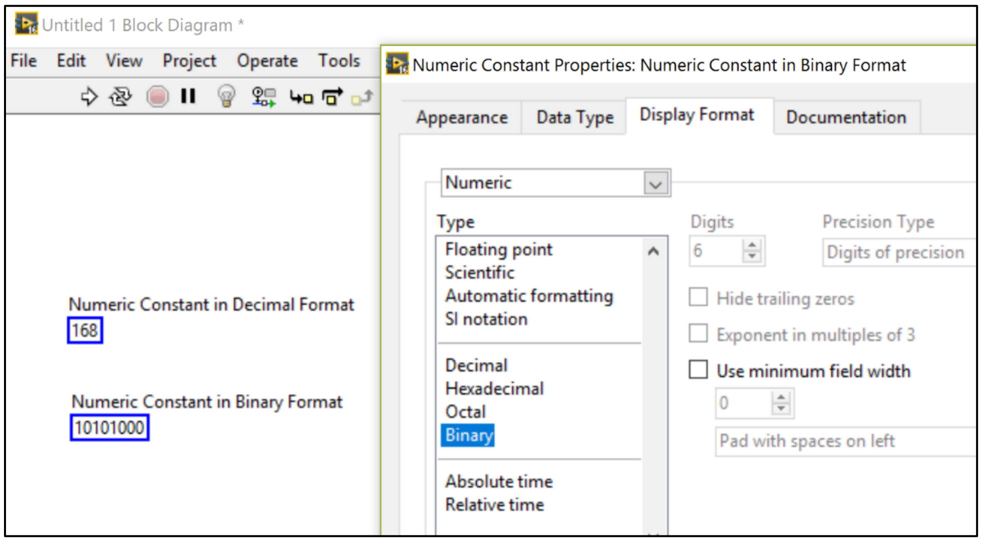
HEX A8

DEC 168

OCT 250

BIN 1010 1000

Figure 1.1: Windows calculator in programming mode verifying $(168)_{10} \rightarrow (1010\ 1000)_2$


Untitled 1 Block Diagram *

Numeric Constant in Decimal Format
168

Numeric Constant in Binary Format
10101000

Numeric Constant Properties: Numeric Constant in Binary Format

Figure 1.2: Verifying $(168)_{10} \rightarrow (1010\ 1000)_2$ using the LabVIEW Block Diagram

Another number format that is frequently used is hexadecimal (Base 16). The reason that Base 16 is a popular alternative to Base 10 in computer applications is due to the fact that Base 16 is divisible by 2 so no bits are "wasted." In other words, it takes 4 bits to represent all 10 of the Base 10 symbols and it also takes 4 bits to represent all 16 of the Base 16 symbols. Therefore, with Base 16 no waste occurs because all 2^4 (or 16) combinations of the four bits can be used to represent 16 different symbols. There are 60% more symbols that are represented with Base 16 as compared to Base 10 when 4 bits are used. In order to keep each of the symbols represented as a single alphanumeric character, the symbol is changed from numbers to letters when going beyond 9 as shown in red font in Table 1.1. Octal (or Base 8) is another option when it is desirable to keep the base divisible by 2 to avoid waste, but use only numbers (and no letters) as symbols. In the Octal number system only digits 0 to 7 are used (as shown in Table 1.1) and only three bits can be represented by a single character. A situation where you would want to avoid using letters is when you are displaying the octal number representation of three bits using a 7 segment display like the one shown in Figure 1.3.

Table 1.1) Comparison of 4 bit numbers in the four most common number formats.

Binary (Base 2)	Decimal (Base 10)	Hexadecimal (Base 16)	Octal (Base 8)
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	10
1001	9	9	11
1010	10	A	12
1011	11	B	13
1100	12	C	14
1101	13	D	15
1110	14	E	16
1111	15	F	17

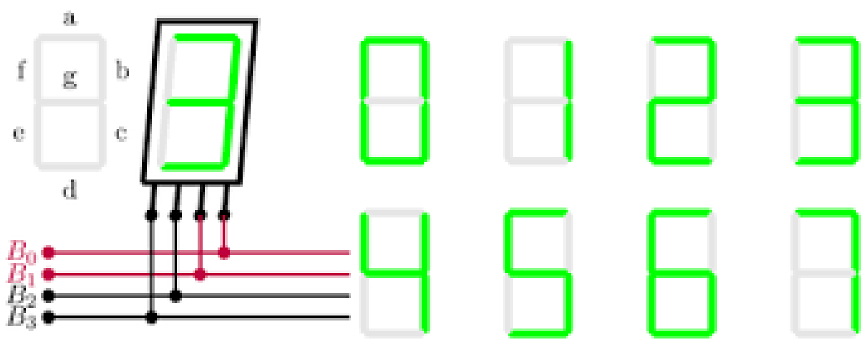


Figure 1.3: A 7-Segment Display to represent an Octal 3-bit symbol (Bit B_3 isn't used). © Germain Gondor. Used under a CC-BY license: http://www.texample.net/tikz/examples/seven-segment-display/

A decimal number that is coded in binary (or BCD) can also be converted to hexadecimal and octal formats. The four formats for the number used in the previous example, along with the conversion process, are shown below.

- **Binary (Base 2) Number Example:** $(1010\ 1000)_2$
- **Decimal (Base 10) Number Example:** $(168)_{10}$
- **Hexadecimal (Base 16) Number Example:** $(A8)_{16}$
- **Octal (Base 8) Number Example:** $(250)_8$

Instead of converting directly from decimal to hexadecimal or octal, it is easiest to first convert from decimal to binary. So, if you want to convert $(168)_{10}$ to the hexadecimal and octal, the first step is to convert it to binary, as shown previously to be equal to $(1010\ 1000)_2$. The following shows how to convert from **binary to hexadecimal**.

1) First, separate the binary number into “nibbles” (A nibble is a group of 4 bits) → $(1010\ 1000)_2$
   - **Note:** *The most significant 4 bits in a byte is called the upper nibble (1010 in this example) and the least significant 4 bits is called the lower nibble (1000 in this example).*
2) Next, list the hexadecimal symbol for each of the 4 bits (see Table 1.1) → 1010 = A, 1000 = 8
3) Finally, list the result as the combination of all hexadecimal symbols → $(A8)_{16}$

The following shows how to convert from **binary to octal**.

1) First, separate the binary number into groups of 3 bits → $(\mathbf{0}10\ 101\ 000)_2$
   - **Note:** *Any time you don’t have enough bits you can always pad the left side with zeros since that will not change the value. The 0 with the red font above is added (or padded) to make three groups of 3 bits.*
2) Next, list the octal symbol for each of the 3 bits (see Table 1.1) → 010 = 2, 101 = 5, 000 = 0
3) Finally, list the result as the combination of all octal symbols → $(250)_8$

Example 1.1) *Convert $(217)_{10}$ to binary, hexadecimal, and octal number formats.*

We will first need to convert the decimal number to binary.

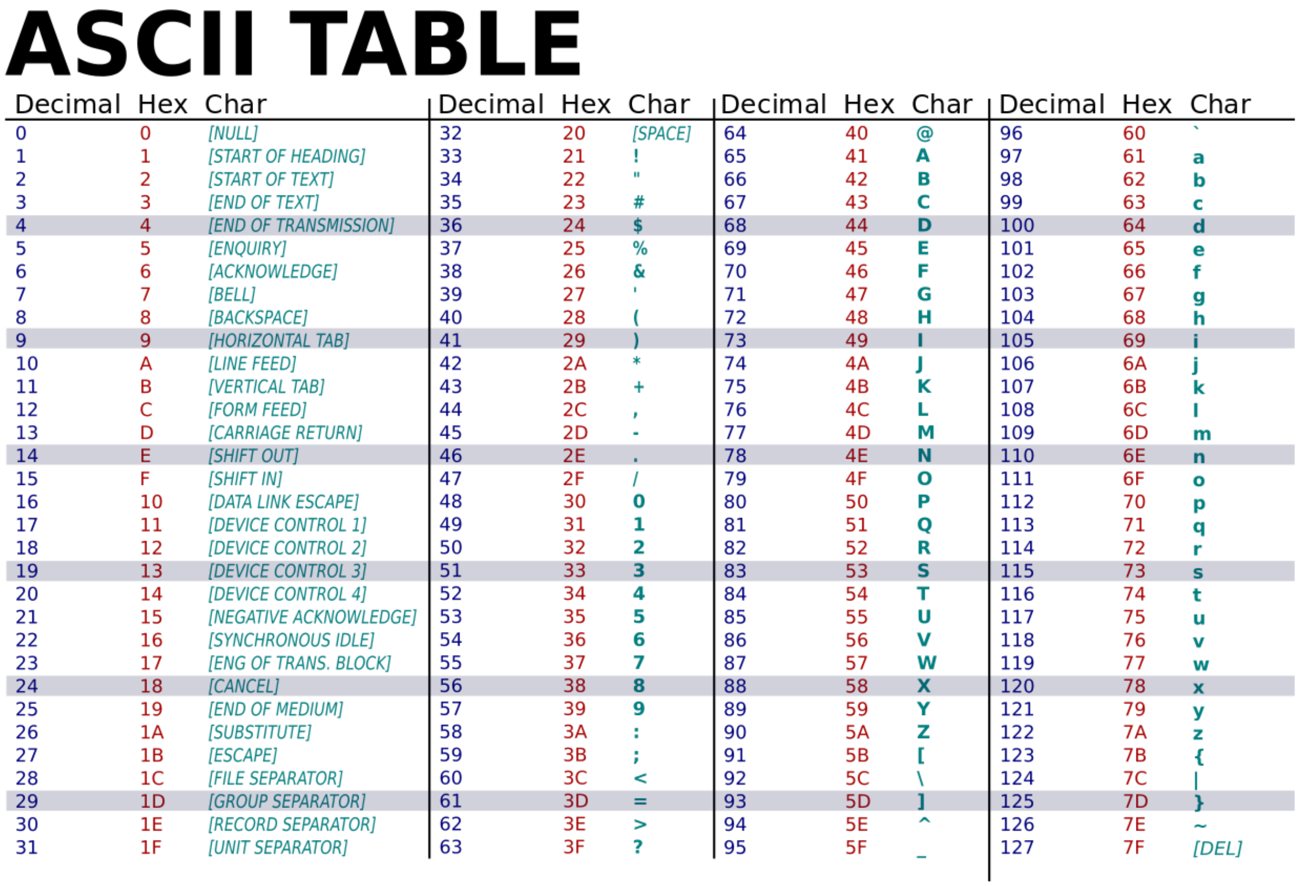
1) 217/128→ Quotient = 1, Remainder = 89 →Divide the decimal number (217) by the weight of the MSB (128)
2) 89/64 → Quotient = 1, Remainder = 25 →Divide the previous remainder (89) by the next bit’s weight (64)
3) 25/32 → Quotient = 0, Remainder = 25 →Divide the previous remainder (25) by the next bit’s weight (32)
4) 25/16 → Quotient = 1, Remainder = 9 →Divide the previous remainder (25) by the next bit’s weight (16)
5) 9/8 → Quotient = 1, Remainder = 1 →Divide the previous remainder (9) by the next bit’s weight (8)
6) 1/4 → Quotient = 0, Remainder = 1 →Divide the previous remainder (1) by the next bit’s weight (4)
7) 1/2 → Quotient = 0, Remainder = 1 →Divide the previous remainder (1) by the next bit’s weight (2)
8) 1/1 → Quotient = 1, Remainder = 0 →Divide the previous remainder (0) by the next bit’s weight (1)

- **Final Result: $(217)_{10}$ → $(1101\ 1001)_2$ → Note:** ***A space separating the upper and lower nibble was included.***

Next, convert the binary (BCD) number to hexadecimal (steps 1 to 3 below) and octal (steps a through c below).

1) First separate the binary number into nibbles (groups of 4 bits) → $(1101\ 1001)_2$
2) Next, list the hexadecimal symbol for each of the 4 bits (see Table 1.1) → 1101 = D, 1001 = 9
3) Finally, list the result as the combination of all **hexadecimal symbols** → $(\mathbf{D9})_{16}$
   a) First separate the binary number into groups of 3 bits → $(\mathbf{0}11\ 011\ 001)_2$
   b) Next, list the octal symbol for each of the 3 bits (see Table 1.1) → 011 = 3, 011 = 3, 001 = 1
   c) Finally, list the result as the combination of all **octal symbols** → $(\mathbf{331})_8$

The previous pages show how ones and zeros are used to represent numbers. This section will show how alphanumeric characters can be represented by ones and zeros so that they can be interpreted by a computer. The process of using a table with different patterns of bits to represent different alphanumeric symbols is often referred to as **character encoding**. The most common method of encoding is called the American Standard Code for Information Interchange (ASCII). The 7-bit ASCII table is shown below. Each alphanumeric character on the table is represented by an 8-bit Hexadecimal number, which contains two hexadecimal symbols.



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 1.4: 7-bit ASCII Table. Public domain: https://simple.wikipedia.org/wiki/ASCII#/media/File:ASCII-Table-wide.svg

Example 1.2) *In the past, it was very popular for the first thing people learned to do on a computer was to create a program that caused the phrase "Hello World" to be displayed on the screen. If "Hello World" was encoded using ASCII, determine the hexadecimal representation of the character string.*

The hexadecimal symbols used for each of the characters is found on the ASCII table in Figure 1.4 as follows:

Characters	"	H	e	l	l	o	space	W	o	r	l	d	"
Hexadecimal	22	48	65	6C	6C	6F	20	57	6F	72	6C	64	22

LabVIEW can also be used to quickly convert character strings to the hexadecimal code from the ASCII table, as shown in the Figure 1.5.

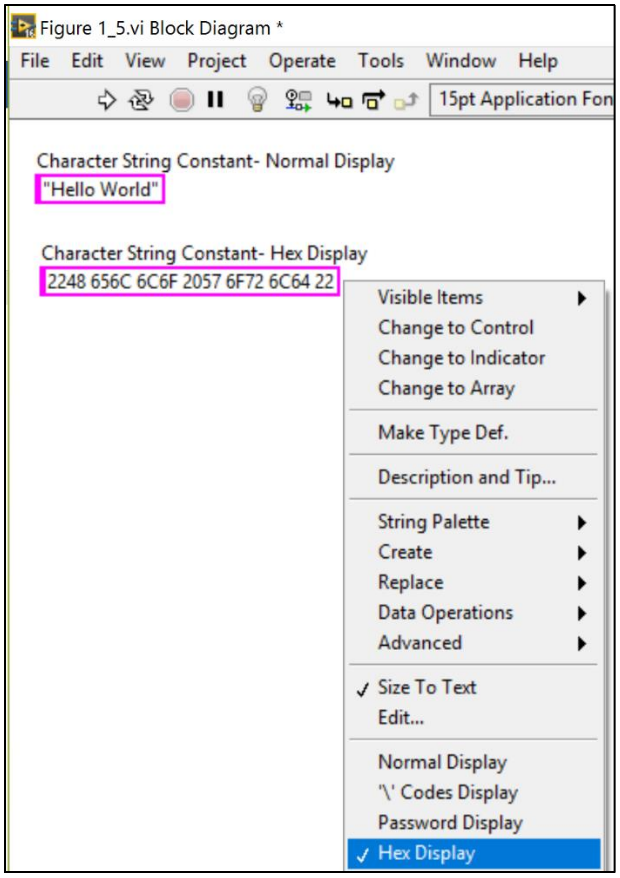


Figure 1_5.vi Block Diagram *

Character String Constant- Normal Display
"Hello World"

Character String Constant- Hex Display
2248 656C 6C6F 2057 6F72 6C64 22

Figure 1.5: LabVIEW used to verify results from Example 1.2

There is an extended character ASCII table that uses 8 bits to allow more characters to be encoded. Every time an extra bit is added to an encoding table, the amount of symbols that can be encoded doubles. For example, with 7 bits 128 symbols can be encoded, but with 8 bits 256 symbols can be encoded. There are also other encoding schemes besides ASCII, but ASCII is by far the most widely used method and the only one that will be discussed here. One situation where a need might arise to use the ASCII table to decode a hexadecimal message is when computer communications is used as discussed in Module 7. Instruments that use serial communication often transmit data to the computer as hexadecimal symbols that correspond to the characters in the ASCII table. One thing that is easy to remember and can be used to partially decode a hexadecimal message, even without having an ASCII table on hand, is the recognition of numbers. Notice that 30 to 39 in the ASCII table correspond to numbers 0 to 9. So, if you see a group of hexadecimal symbols streaming across the computer screen remember that tip and you can quickly decode all of the numbers in the message.

Module 1 – References and Links

The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[1] More information about FETs and how they are used to create logic gates can be found at: https://www.allaboutcircuits.com/textbook/digital/chpt-3/cmos-gate-circuitry/
[2] Binary Coded Decimal (BCD) info: https://en.wikipedia.org/wiki/Binary-coded_decimal
[3] Extended character ASCII table: https://en.wikipedia.org/wiki/Extended_ASCII

Module 2 – Digital Logic

Module 1 introduced the binary number system, which is composed of ones and zeros. This module will discuss digital logic gates that have inputs and outputs that are at a digital high level (1 or true) or low level (0 of false).

Note: *We will assume "active high" logic, where a one (1) is the number associated with the "true" state or a "high" digital logic level and zero (0) is associated with the "false" state or the "low" digital logic level.*

When solving a digital logic circuit (which is a combination of logic gates) mathematically it is often referred to as "Boolean Algebra." In hardware, Boolean operations are performed by physical components called digital logic gates. Each input and output of these gates are at a voltage level that correspond to a one or zero. Historically, the most common "**logic level**" was the 5V logic level, where 5 Volts corresponds to a one (or true state) and 0 Volts corresponds to a zero (or false state).

Note: *The **logic level** is a term that is often called the "supply voltage" because it is the voltage level of the power supply (or power rail) of the circuit and the maximum possible input or output voltage of each gate. In hardware, there is a range of voltages that correspond to high or low logic levels, as shown in Table 2.1.*

The 5V logic level is used as the standard for Bipolar Junction Transistors (BJT) that are used to make logic gates. In the past, BJTs were the primary type of transistors used to create logic gates and the 0 to 5 Volt logic level standard is called Transistor-Transistor Logic (TTL). In more recent times, the Field Effect Transistors (FET) have taken over as the type of transistor used to make logic gates. For FETs, the voltage level is variable and is being continually reduced as the manufacturing processes are improved that allow for smaller FET channel lengths. Table 2.1 shows the logic level for TTL and CMOS (Complementary Metal Oxide Semiconductor) as well as the transition voltages that separate the high (true state) and low (false state) logic levels for each.

Note: *CMOS is the most common type of FET. More information about FETs and how they are used to create logic gates can be found at: https://www.allaboutcircuits.com/textbook/digital/chpt-3/cmos-gate-circuitry/*

Table 2.1) The two most common types of logic levels. Sources: https://en.wikipedia.org/wiki/Logic_level and https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/

Transistor Type	Low Voltage Range	High Voltage Range	Supply Voltage Level
TTL	0 V to 0.8 V	2 V to V_{CC}	V_{CC} = 5 V ±10%
CMOS	0 V to ⅓ V_{DD}	⅔ V_{DD} to V_{DD}	V_{DD} is adjustable

Digital logic circuits (also called combinational logic circuits) at their lowest level are created using different combinations of the following three gates (all gates will be listed in CAPS in this eBook so they stand out).

- **NOT** Gate (also called an inverter) – The **output is only true** when the input is NOT true (i.e false).
- **AND** Gate – For a 2 input AND gate, the **output is only true** when both input 1 AND input 2 are true.
- **OR** Gate – For a 2 input OR gate, the **output is only true** when either input 1 OR input 2 are true.

The NOT gate always has one input and one output, but the AND gate and OR gate can have as many inputs as desired. The verbiage in the rules when the output is true for AND gates and OR gates must be changed slightly when there are more than two inputs. The following definitions are not as easy to remember as the previous ones shown in the bullets above, but they are true for all possible numbers of inputs (2 or greater).

- **AND** Gate – For an AND gate with X inputs, the **output is only true** when all X inputs are true.
- **OR** Gate – For an OR gate with X inputs, the **output is true** when any of the X inputs are true.

Section 2.1 – NOT, AND, OR, XOR, NAND, NOR, and XNOR Gate Overview

Truth tables are a common way to describe the operation of either a gate or a digital logic circuit. It is simply a table that shows the Boolean values (0 or 1) of all the inputs and outputs. A truth table can be made for a single gate or for any digital logic circuit. The number of inputs determines how many unique states (or different input combinations) there are in a circuit. **The number of states is equal to 2^X, where X = the number of inputs.** Figure 2.1 shows the NOT gate truth table and common symbols that are used to represent the NOT gate.

Input	Output
0	1
1	0

(a)

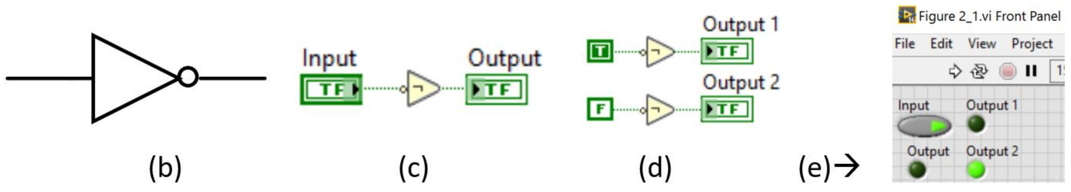
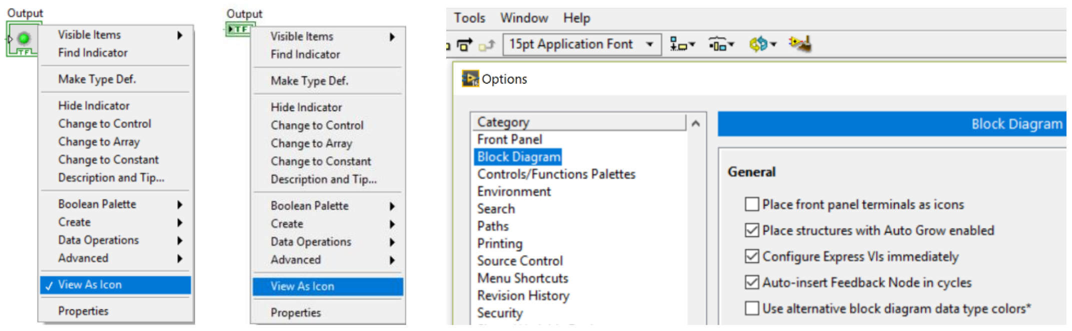


Figure 2.1: (a) NOT gate truth table. (b) Multisim NOT gate symbol. (c) LabVIEW NOT gate symbol with a "Control" (or adjustable) Input. (d) LabVIEW NOT gate symbols with "Constant" (or fixed) inputs. (e) LabVIEW Front panel for the Block Diagram code in (c) and (d) with the "Input" Control set as True on the Front Panel.

Figure 2.1 c & d shows LabVIEW NOT gate symbols. For "**Boolean**" data type, all wires and icons are **green** on the Block Diagram (*LabVIEW's programming environment*). An output in LabVIEW is called an "indicator" and will have its value changed on the Front Panel (*LabVIEW's User Interface*) when the program is executed. For Boolean outputs, the indicator on the front panel is an LED that will light up when its value is "True" as shown in Figure 2.1e. Figure 2.1c shows an input called a "**control**" (the control can have its value changed on the Front Panel). Notice in Figure 2.1c that a "control" contains a filled-in outside edge on its icon in the Block Diagram.

Note: *The controls and indicators in Figure 2.1 and throughout this entire eBook are displayed in the standard style with smaller footprints. The default symbol view in LabVIEW is called "Icon View" and makes the controls and indicators much larger as shown in Figure 2.2a. In order to have more room on the Block Diagram, it is recommended to change the default to standard view by unchecking the top checkbox (shown in Figure 2.2c).*



- [ ] Place front panel terminals as icons
- [x] Place structures with Auto Grow enabled
- [x] Configure Express VIs immediately
- [x] Auto-insert Feedback Node in cycles
- [ ] Use alternative block diagram data type colors*

(a) (b) (c)

Figure 2.2: (a) Indicator in "Icon" View. (b) Indicator in standard view. (c) Uncheck the box under "Place front panel terminals as icons" in the LabVIEW Menu under ->Tools->Options->Block Diagram

Figure 2.1d shows Boolean "**constant**" inputs, which are "hard-coded" variables than cannot be changed on the Front Panel. The small circle in the NOT gate symbol signifies there is an inversion (i.e. 1 becomes 0 or a 0 becomes 1). The circle is usually placed after the triangle as shown in Figure 2.1b, but in LabVIEW it is placed before the triangle as shown in Figure 2.1c. Any time a gate symbol includes a small circle (e.g. NAND, NOR, XNOR) it is an indicator that there is an inversion. One good thing about LabVIEW symbols for logic gates is that they also include the text symbol that is used for the gate inside the outline of the component. For Boolean algebra math problems, only text symbols are used.

Figure 2.3 and Figure 2.4 show truth tables and common symbols that are used for the AND gate and OR gate, respectively. A digital logic circuit that can be used to create an AND gate from an OR gate and vice-versa is also shown for each in Figures 2.3d and 2.4d. This is useful to know if only one of the types of gates is available.

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

(a)

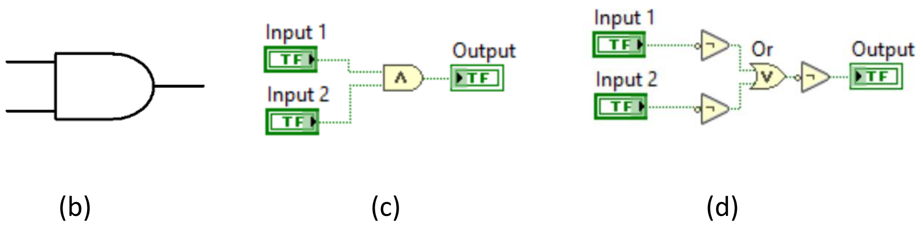


Figure 2.3: (a) Truth table for the 2 input AND gate. (b) Multisim AND gate symbol. (c) LabVIEW AND gate symbol. (d) Digital logic circuit that can be used to create an AND gate from NOT gates and an OR gate.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

(a)

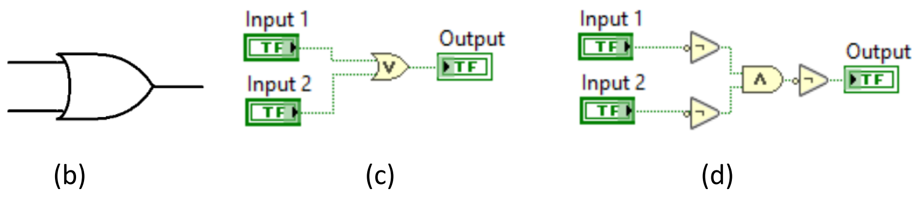
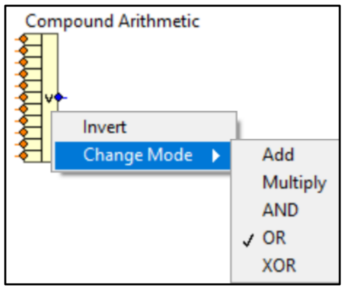


Figure 2.4: (a) Truth table for the 2 input OR gate. (b) Multisim OR gate symbol. (c) LabVIEW OR gate symbol. (d) Digital logic circuit that can be used to create an OR gate from NOT gates and an AND gate.

Note: *The* ***inputs of a truth table are always listed in increasing decimal order*** *(00 = 0, 01 = 1, 10 = 2, 11 = 3).*



Multisim has several different AND gate and OR gate parts that go up to 8 inputs. However, in LabVIEW all AND gate and OR gate parts with more than 2 inputs are implemented with a block called "Compound Arithmetic" (as shown in Figure 2.5). A couple of examples of these are shown in Figure 2.6. The Compound Arithmetic block in LabVIEW allows the number of inputs to be changed by re-sizing the block and it also allows other mathematical operations to be selected by clicking on the text symbol and selecting Change Mode as shown in Figure 2.5.

Figure 2.5: LabVIEW OR gate with 11 inputs (Note: *If "Invert" was selected it would change the OR to a NOR*).

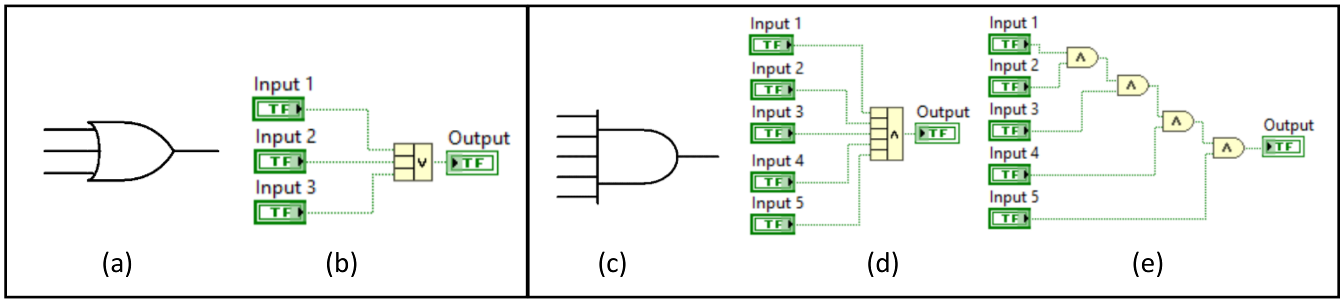


Figure 2.6: (a) Multisim OR gate with 3 inputs. (b) LabVIEW OR gate implementation using the Compound Arithmetic block with 3 inputs. (c) Multisim AND gate with 5 inputs. (d) LabVIEW AND gate implementation using the Compound Arithmetic block with 5 inputs. (e) 5-input AND gate implemented with 2-input gates.

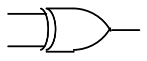
Another commonly used gate is the exclusive OR (also called XOR or EOR) gate. This gate can be defined as:

- **XOR** Gate – For an XOR gate with X inputs, the **output is only true** if any of the X inputs are different.

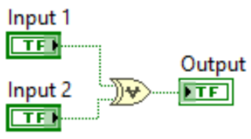
The way to remember the XOR definition is to think of an X as the mark that is often given on a paper if you have an answer that is different from the solution. In this way, rows 2 and 3 of the truth table in Figure 2.6a are the only two states that produce a true (or 1) output because these are the only two states that have inputs that are different (i.e. Input 1 ≠ Input 2). For an XOR gate with more than 2 inputs, the output is a true if any of the inputs are different. Conversely, the output of an XOR gate is false only if all of the inputs have the same value.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

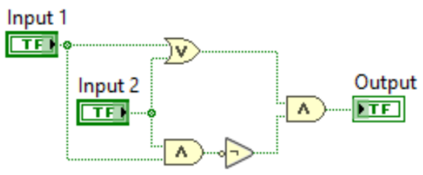
(a)



(b)



(c)



(d)

Figure 2.7: (a) Truth table for the 2 input XOR gate. (b) Multisim XOR gate symbol. (c) LabVIEW XOR gate symbol. (d) Digital logic circuit that can be used to create an XOR gate from AND, OR, and NOT gates.

Note: *Another text symbol used for XOR, instead of the OR symbol with a line through it, is a circle with a + → ⊕.*

The AND, OR, and XOR gates could be considered "active high" gates because their output is true (or high) when the inputs correspond to their name. For the 2 input versions of the gates, the **active high** gate definitions are:

- AND Gate – The output is only **true** when both input 1 AND input 2 are true.
- OR Gate – The output is only **true** when either input 1 OR input 2 are true.
- XOR Gate – The output is only **true** when input 1 and input 2 are different values.

The NAND, NOR, and XNOR gates are the "active low" versions of AND, OR, and XOR because they have the opposite outputs. The **active low** gate definitions are shown below. The CMOS circuit implementations of these gates are shown at this link → https://www.allaboutcircuits.com/textbook/digital/chpt-3/cmos-gate-circuitry/

- NAND Gate – The output is only **false** when both input 1 AND input 2 are true.
- NOR Gate – The output is only **false** when either input 1 OR input 2 are true.
- XNOR Gate – The output is only **false** when input 1 and input 2 are different values.

Table 2.2) Truth table for the 6 types of 2-input digital logic gates.

Input 1	Input 2	**NAND** Output	AND Output	**NOR** Output	OR Output	**XNOR** Output	XOR Output
0	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1
1	0	1	0	0	1	0	1
1	1	0	1	0	1	1	0

The three active low gates (NAND, NOR, XNOR) can be created from their corresponding active high gates by adding a NOT gate on the output as shown in Figure 2.8.

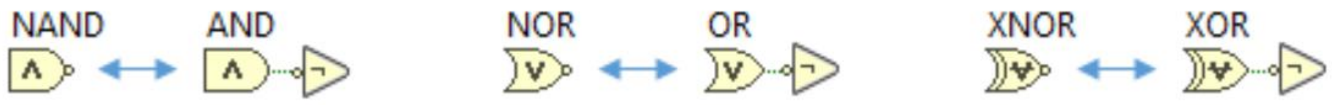
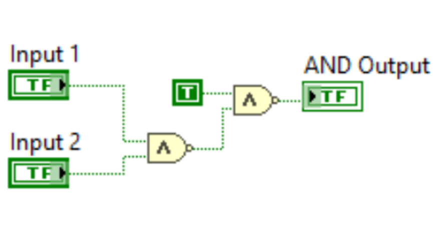
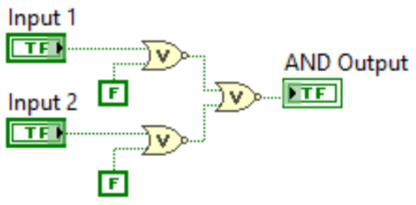


Figure 2.8: (a) NOR = OR gate with NOT. (b) NAND = AND gate with NOT. (c) XNOR = XOR gate with NOT.

NAND and NOR gates are especially useful because all gate types can be made using a combination of only one of these type of gates. For example, an AND gate also needs to have a NOT gate available if it were to be used to create a NAND gate, but the following figure shows how to make an AND gate from only NAND gates or only NOR gates. See the following link for more information about how to create other types of gates from only NAND or NOR gates: https://www.allaboutcircuits.com/textbook/digital/chpt-3/gate-universality/

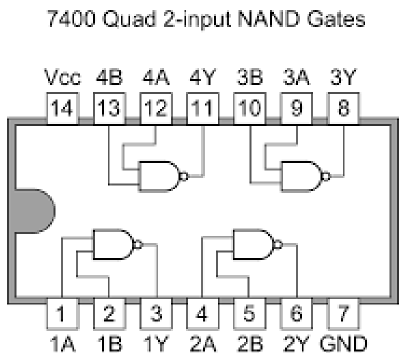


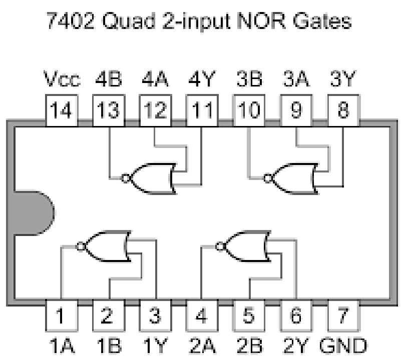


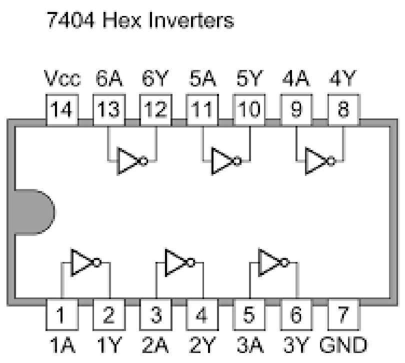
(a) (b)

Figure 2.9: (a) AND gate created from only NAND gates. (b) AND gate created from only NOR gates.

If the circuits in Figure 2.9 were implemented in hardware a NAND or NOR gate integrated circuit (IC) could be used that included multiple gates. Being able to use only one type of IC to create all of the possible gate types can reduce cost and add more flexibility in the design of digital circuits. Figure 2.10 shows some popular 7400 series digital ICs. The following link shows more details about the history of this popular logic family and a list of the 7400 series part numbers. https://en.wikipedia.org/wiki/List_of_7400_series_integrated_circuits







(a) (b) (c)

Figure 2.10: 7400 series ICs. (a) Quad NAND gate IC. © Tosaka. Used under license CC BY: https://commons.wikimedia.org/wiki/File:7400_Quad_2-input_NAND_Gates.PNG **(b) Quad NOR gate IC.** © Tosaka. Used under CC BY-SA license: https://commons.wikimedia.org/wiki/File:7402_Quad_2-input_NOR_Gates.PNG **(c) Hex inverter IC.** © Tosaka. Used under a CC-BY license : https://commons.wikimedia.org/wiki/File:7404_Hex_Inverters.PNG

Section 2.2 – Digital Logic Circuit Examples

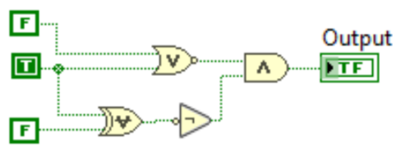
The goal of this eBook is to provide the most important aspects of numerous topics without getting bogged down into time consuming details that would not be of interest to the majority of the readers. If it is desired to go deeper into digital logic and Boolean algebra concepts, https://www.allaboutcircuits.com has an open source digital eBook that can help fill in the gaps. → https://www.allaboutcircuits.com/textbook/digital/

This section will show how to solve and design different types of digital logic circuits. The focus will be on the implementation of digital logic into software, but the concepts could also be applied in building the digital logic circuits with hardware using logic gate ICs, like those previously discussed. LabVIEW is used for all the examples in this book, but digital logic can be implemented in many types of software and even in programs like Excel.

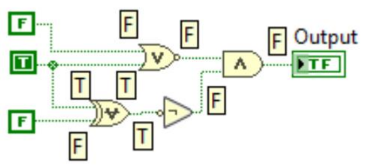
The first three examples are the simplest type of digital problems where the inputs have true or false constant inputs. On the right side of each example the solution is shown with a T or F labeled on the output of every gate.

Note: *If a wire splits and goes in multiple directions, the T or F signal on that wire goes to each gate input (e.g. In Example 2.1, the True constant splits up and goes to the NOR and XOR gates). Also, when a split occurs lines frequently cross. If there is a line cross, the wires aren't connected unless there is a green dot touching the intersection of the lines (e.g. In Example 2.2 the top False constant splits and crosses the True constant wire and connects to the input of the OR gate. Since there is no green dot at the intersection, there is not a connection).*

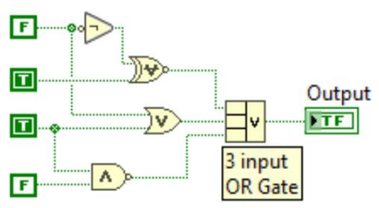
Example 2.1) *What is the output of the following digital logic circuit?*



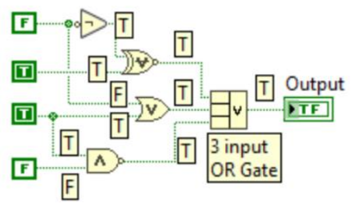
Solution →



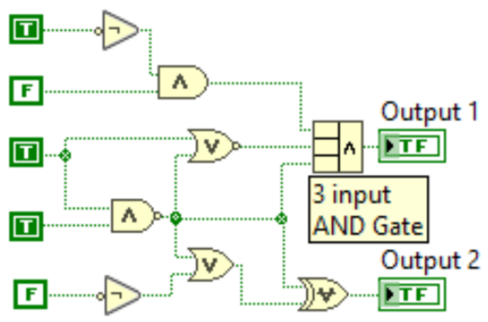
Example 2.2) *What is the output of the following digital logic circuit?*



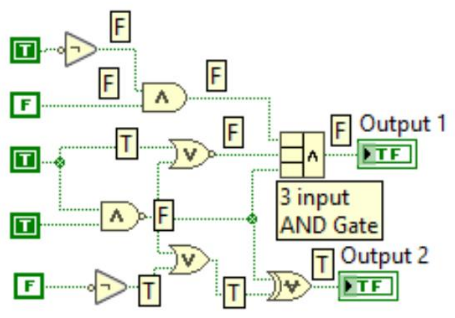
Solution →



Example 2.3) *What is the output of the following digital logic circuit?*

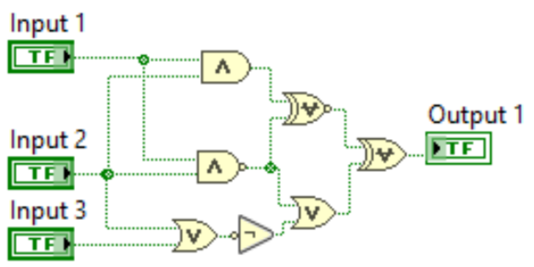


Solution →



The next two examples involve creating truth tables for digital logic circuits with inputs that are "controls" (i.e. inputs that can be changed to True or False on the Front Panel). In order to solve these circuits, the input rows are populated in increasing decimal order and the outputs are solved using the techniques in Examples 2.1 - 2.3.

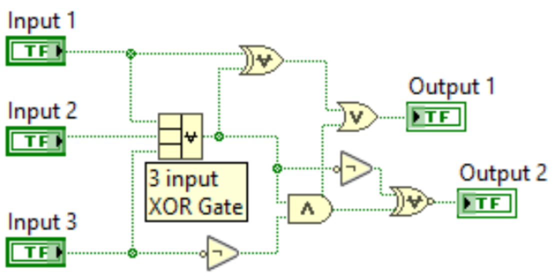
Example 2.4) *Create the truth table for the following digital logic circuit.*



Solution →

Input 1	**Input 2**	**Input 3**	**Output**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Example 2.5) *In this example, assume that only the cells of the truth table that are shaded yellow are unknown. To determine these unknown values, the inputs need to be determined by making sure each row has a decimal equivalent value 1 higher than the previous row (i.e. 010 = 2, 011 = 3, etc.). For the output fields, determine the values for each cell as a separate problem using the corresponding inputs of that row, as done in Examples 2.1-4.*



Solution →

Input 1	Input 2	Input 3	Output 1	Output 2
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Note: *The XOR symbol (OR symbol with a line through it) is really hard to distinguish from the OR gate on the Compound Arithmetic block. Recall, another text symbol used for the XOR gate is a circle with a + inside it → ⊕. Unfortunately, LabVIEW doesn't use the circle with a plus sign text symbol for the XOR gate.*

Section 2.3 – Bitwise Operations

Bitwise operations are another important concept that deal with logic gates. These operations require each bit in a binary number to be operated on using the logic gate truth table. For example, if you want to perform a bitwise AND on two bytes it would be completed by starting with the LSB and taking the AND of the two LSB bits and proceeding to the left and repeating the process until the MSB is reached. Example 2.6 shows this process.

Example 2.6) *Determine the bitwise AND for Byte 1 and Byte 2.*

Byte 1	0011 0101
Byte 2	1010 1100
Bitwise AND of Byte 1 and Byte 2	0010 0100

As each bit is "**AND**ed" (AND text symbol = ∧) the results in row 3 shows that the **only bits that remain a 1** are **bits in positions where both** Byte 1 and Byte 2 were **equal to 1**. The 8 steps to this problem are shown below, starting with the LSB.

Step 1. Byte 1_bit0 ANDed with Byte2_bit0 = 1 ∧ 0 = 0 (This is the LSB)
Step 2. Byte 1_bit1 ANDed with Byte2_bit1 = 0 ∧ 0 = 0
Step 3. Byte 1_bit2 ANDed with Byte2_bit2 = 1 ∧ 1 = 1
Step 4. Byte 1_bit3 ANDed with Byte2_bit3 = 0 ∧ 1 = 0
Step 5. Byte 1_bit4 ANDed with Byte2_bit4 = 1 ∧ 0 = 0

Step 6. Byte 1_bit5 ANDed with Byte2_bit5 = 1 ∧ 1 = 1
Step 7. Byte 1_bit6 ANDed with Byte2_bit6 = 0 ∧ 0 = 0
Step 8. Byte 1_bit7 ANDed with Byte2_bit7 = 0 ∧ 1 = 0 (This is the MSB)

Example 2.7) *Determine the bitwise AND, NAND, OR, NOR, XOR, and XNOR for Bytes 3 and Byte 4.*

Byte 3	Byte 4	AND	NAND	OR	NOR	XOR	XNOR
0111 0011	1100 0001	0100 0001	1011 1110	1111 0011	0000 1100	1011 0010	0100 1101

For the bitwise OR symbol, each bit is a 1 except those bit positions where both Byte 3 and 4 were equal to zero (bits 2 and 3). For the bitwise **XOR**, a **1 bit indicates** that at the corresponding bit position Bytes 3 and 4 had **different values** (bits 1, 4, 5, and 7). The active low gates (NAND, NOR, and XNOR) bitwise values are equal to their corresponding active high gate (AND, OR, and XOR) bitwise values with all bits flipped (or inverted).

"Bit Masking" is one situation where bitwise operations are used on a data packet (a sequence of ones and zeros). The following are the four common types of bit masking. Go to the following link for more details about bit masking: https://en.wikipedia.org/wiki/Mask_(computing)

- **Setting the value of specific bits to 1 using the bitwise OR function**. For example, if you wanted to set bits 0 and 1 of Byte 4 in Example 2.7 to 1, but not change the values of bits 2 through 7 then you would bitwise OR Byte 4 (1100 00**01**) with the "bitmask" of 0000 00**11** to get a result of 1100 00**11**.
- **Setting the value of specific bits to 0 using the bitwise AND function**. For example, if you wanted to set bits 0 and 1 of Byte 4 in Example 2.7 to 0, but not change the values of bits 2 through 7 then you would bitwise AND Byte 4 (1100 00**01**) with the "bitmask" of 1111 11**00** to get a result of 1100 00**00**.
- **Determine the value of bits using the bitwise AND function**. For example, if you wanted to see what the values of bits 0 and 1 of Byte 4 are in Example 2.7 then you would bitwise AND Byte 4 (1100 00**01**) with the "bitmask" of 0000 00**11** to get a result 0000 00**01**. By setting the bitmask of bits 2 to 7 to zero the bitwise AND of those bits are forced to zero and bits 0 and 1 of Byte 4 are set to the original value.
- **Inverting specific bits using the bitwise XOR function**. For example, if you wanted to invert (i.e. flip a 1 to 0 or a 0 to 1) bits 0 and 1 of Byte 4 in Example 2.7, but not change the values of bits 2 through 7 then you would bitwise XOR Byte 4 (1100 00**01**) with the "bitmask" of 0000 00**11** to get a result of 1100 00**10**.

Another bitwise operation that is useful is using the bitwise XOR operation to see if two data packets are the same. For example, if a 16 bit checksum of a data packet is computed to be equal to 1010 1010 1010 1010 before it was transmitted, but when it was received the checksum was computed to be equal to 1110 1010 1101 1111. The bitwise XOR of the two 16 bit numbers is equal to 0100 0000 0111 0101. Since the result of the bitwise XOR is not equal to zero, then it is known that the two checksums are different and either bits were changed in the transmission of the data packet, or the sender or receiver made an error in calculating the checksum. Additional discussion on detecting bit errors and other data transmission topics are discussed in Module 7 – Computer Communications. More information about computing checksums and common algorithms that are used are found at the following link: https://en.wikipedia.org/wiki/Checksum

Another common use of bitwise operations is used in keeping track of the status of a processor. Specific bits (called "flags") are dedicated to keeping track of the status of a certain operation or interrupt. For example, if two numbers are added together and the result is too large to fit in the number of bits that are allocated for it then an overflow flag would be set to 1 by using the bitwise OR function (shown in the first bullet shown above). There is more information about flags at the following link: https://en.wikipedia.org/wiki/Status_register

Section 2.4 – Thresholding

Thresholding is often done to convert an analog number to a digital number. For example, if a 4-AA battery cage with ~ 6V is used as the power source in a CMOS circuit, then according to Table 2.1 the V_{DD} value would be set to 6V, the low threshold would be set to 2V, and the high threshold would be set to 4V. A 2V threshold could be applied to the analog value in this circuit so that any signal that is less than 2V is set to 0 (or False). Additionally, a 4V threshold could be set so that any signal that is greater than 4V is set to 1 (or True). Anything between 2V and 4V would be considered to be in an indeterminate state for CMOS. LabVIEW has tools (Shown in Figure 2.11) that allows many different types of comparisons to be made.

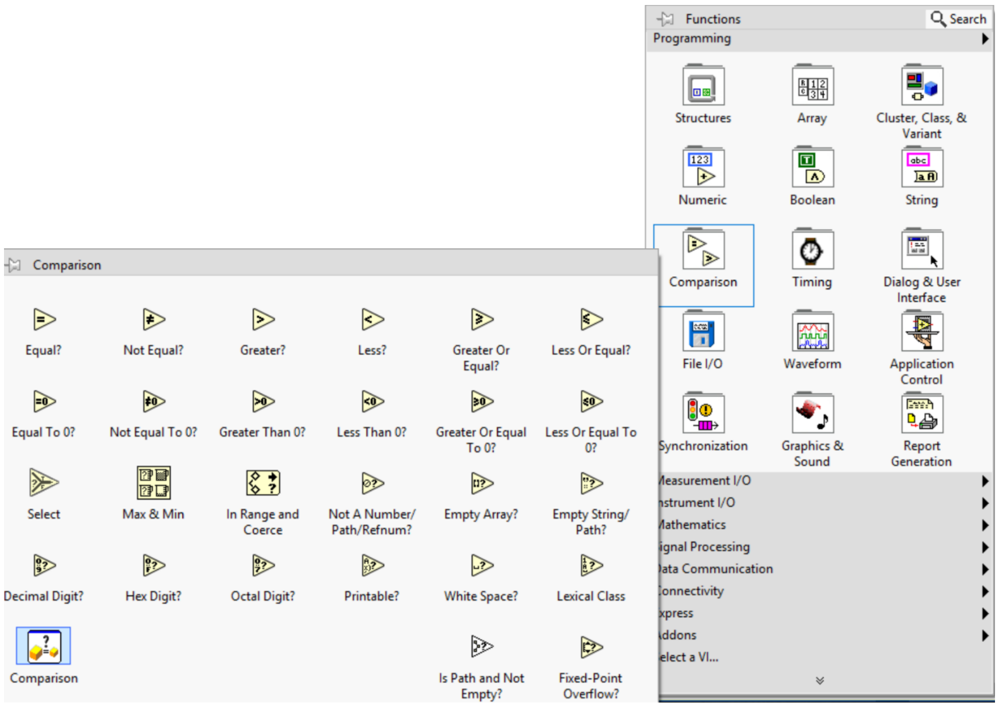


Figure 2.11: LabVIEW Comparison Tools Menu

In hardware, a threshold is often made using a comparator circuit (such as the one shown in Figure 2.12 that uses the LM324 OpAmp). In this CMOS logic level indicator circuit, a red LED is turned on when the input value is less than 2V (**Vin1** below) and a green LED is turned on if the input value is greater than 4V (**Vin2** below).

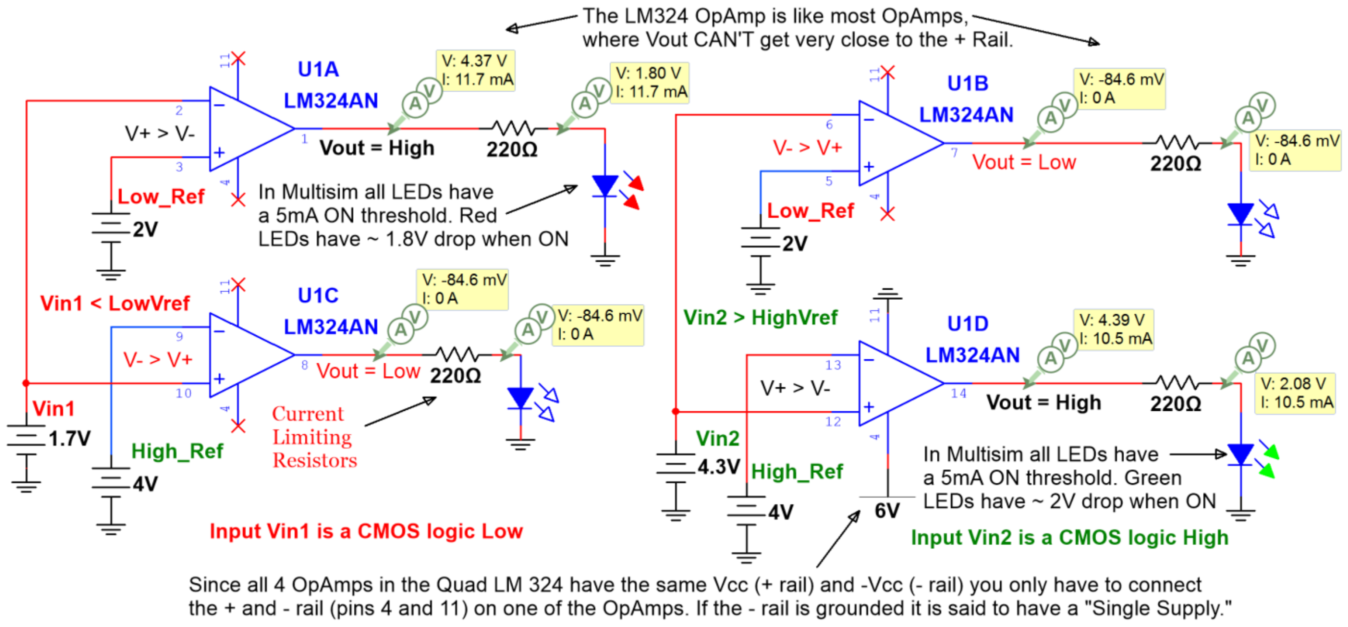
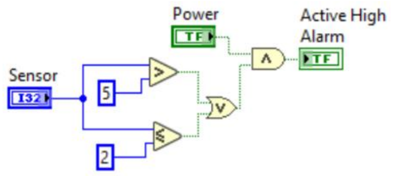

The LM324 OpAmp is like most OpAmps, where Vout CAN'T get very close to the + Rail.
U1A
LM324AN
V: 4.37 V
I: 11.7 mA
V: 1.80 V
I: 11.7 mA
V+ > V-
Vout = High
220Ω
Low_Ref
2V
In Multisim all LEDs have a 5mA ON threshold. Red LEDs have ~ 1.8V drop when ON
Vin1 < LowVref
U1C
LM324AN
V: -84.6 mV
I: 0 A
V: -84.6 mV
I: 0 A
V- > V+
Vout = Low
220Ω
Vin1
1.7V
High_Ref
4V
Current Limiting Resistors
Input Vin1 is a CMOS logic Low
U1B
LM324AN
V: -84.6 mV
I: 0 A
V: -84.6 mV
I: 0 A
V- > V+
Vout = Low
220Ω
Low_Ref
2V
Vin2 > HighVref
U1D
LM324AN
V: 4.39 V
I: 10.5 mA
V: 2.08 V
I: 10.5 mA
V+ > V-
Vout = High
220Ω
Vin2
4.3V
High_Ref
4V
6V
In Multisim all LEDs have a 5mA ON threshold. Green LEDs have ~ 2V drop when ON
Input Vin2 is a CMOS logic High
Since all 4 OpAmps in the Quad LM 324 have the same Vcc (+ rail) and -Vcc (- rail) you only have to connect the + and - rail (pins 4 and 11) on one of the OpAmps. If the - rail is grounded it is said to have a "Single Supply."

Figure 2.12: CMOS Logic level indicator circuit using a LM324 Quad Operational Amplifier IC as a comparator.

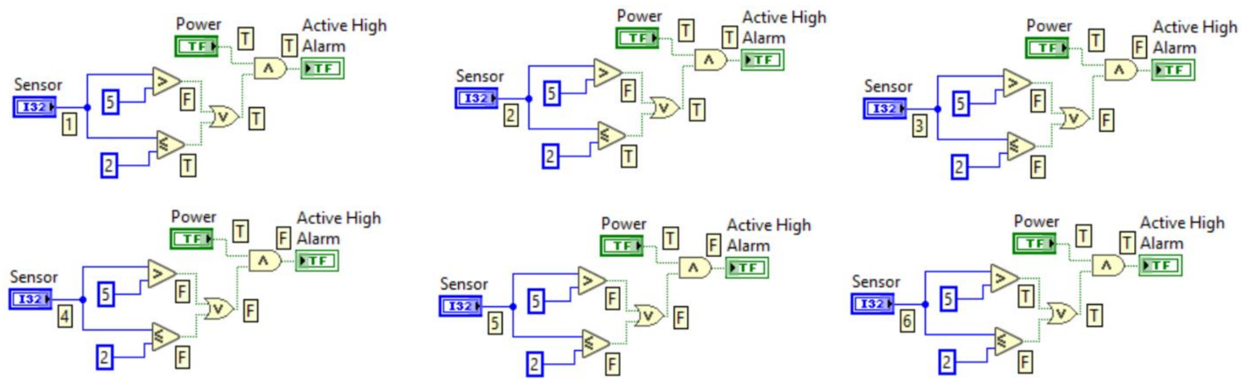
Many digital Logic ICs such as the SN74AHC132N Quad NAND gate have built-in Schmitt Trigger (a comparator with hysteresis) inputs so that the analog input is internally converted to digital (i.e 0 Volts or V_{DD}) [15]. This elimiates the need for a separate comparator circuit, like the one shown in Figure 2.12. More information about Schmitt triggers is shown in reference 16. Module 7 of the Davis AC Circuits Book covers Operational Amplifiers (Op Amps) in more detail and provides examples of circuits that use integrated circuits that are specially designed to serve as comparators [17]. Op Amps like the LM324 can be used as a comparator (as shown in Figure 2.12), but they are designed to be used as amplifiers where a resistor is connected between Vout and Vin(-) to serve as feedback [18].

The rest of this section will focus on the thresholding of digital logic circuits that are implemented in software using the LabVIEW tools shown in Figure 2.11. The CMOS indicator circuit in Figure 2.12 could be implemented in LabVIEW using a similar method to what is shown in Example 2.8. In this section, the digital logic example problems will have analog inputs (numeric values) instead of digital inputs (True or False) so they need to be thresholded (using a comparison LabVIEW block) to be converted to digital.

Example 2.8) *For the following digital logic circuit with the* *<u>Power input set to True</u>*, *determine whether the alarm will be ON or OFF for Sensor input values of [1, 2, 3, 4, 5, 6]. This bracket notation means that different sensor values are input into the circuit at different times to determine what outputs occur.*

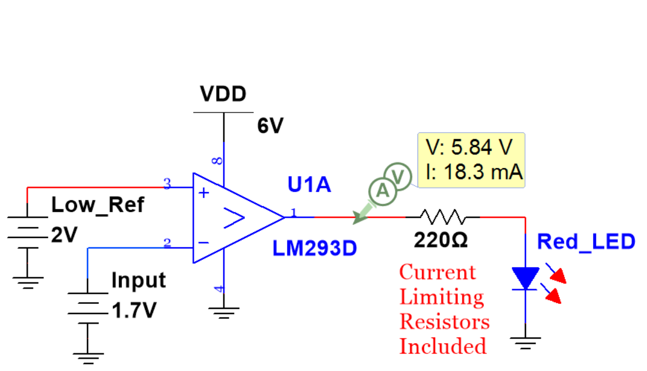


The following shows the results for all of the Sensor inputs. The corresponding outputs for each Sensor input are [On, On, Off, Off, Off, On]. If the <u>Power input was changed to False</u>, the alarm would be OFF for all sensor values.

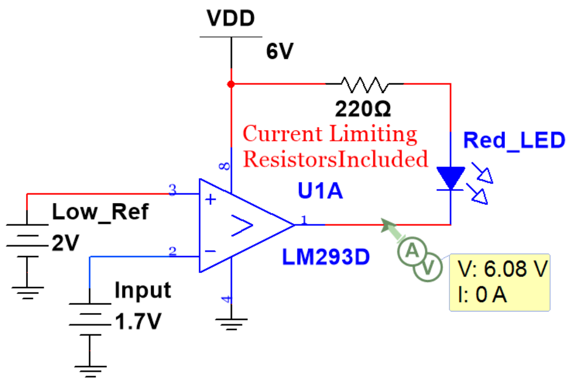


<u>**Note:**</u> *The* ***<u>blue</u>*** *controls, constants, and wires in LabVIEW represent* ***<u>decimal (or integer) data type</u>***.

The output of Example 2.8 has an <u>**Active High**</u> alarm, which means it is energized (or <u>turns on</u>) when the <u>input is true</u>. An <u>**Active Low**</u> device would act in the opposite manner (<u>turns off</u> when the <u>input is true</u>). If a device is connected between the output of a device and ground it would act as an active high device, but if a device is connected between a power rail (V_{DD}) and the output of a device then it would act as an active low device. An example of this is shown in Figure 2.13, where circuit (a) has an LED that is connected between the output of a comparator and ground and it turns ON when the comparator output is a logic high (or True) and circuit (b) has an LED that is connected between the power rail and the output of a comparator and it turns OFF when the comparator output is a logic high (or True). When the comparator output is a logic low (or False) the active high circuit (a) would turn OFF and the active low circuit (b) would turn ON.



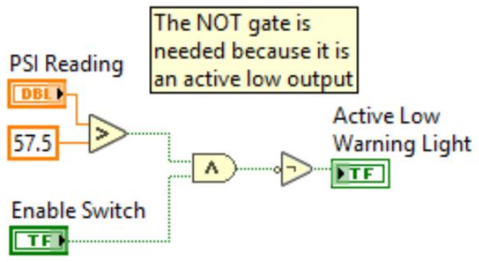
(a)



(b)

Figure 2.13: (a) LED in <u>Active High</u> Configuration (LED turns On when the comparator has a logic high output). (b) LED in <u>Active Low</u> Configuration (LED turns Off when the comparator has a logic high output).

Example 2.9) *Create a logic circuit that turns an active low warning light on when a pressure sensor reading is greater than 57.5 psi and the enable switch is true.*



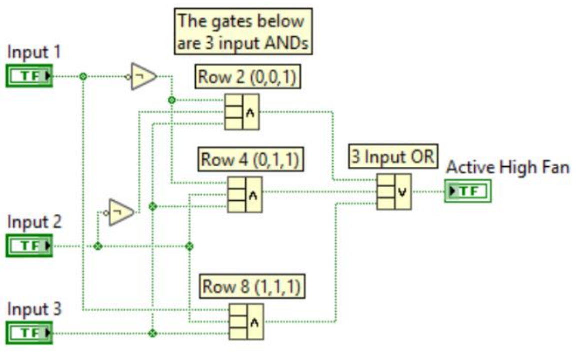
<u>**Note:**</u> *The <u>orange</u> controls, constants, and wires in LabVIEW represent <u>**fractional data type**</u>.*

Section 2.5 – Creating a Logic Circuit from a Truth Table

This section shows how to create a truth table from a list of specifications and then convert the truth table into a digital logic circuit using the "**Isolation-OR method**". The Isolation-OR method <u>always works</u>, but often uses more gates than are necessary. Reducing the gate count is important in hardware, but not in software. The Isolation-OR method is as follows: <u>OR together each row that has a true output</u> after <u>all zero inputs in that row are inverted to be ones</u> and <u>ANDed together with the other true inputs</u>. Examples 2.10 - 2.11 show this method.

Example 2.10) *Create the truth table and logic circuit for a system that has the following specifications. An active high fan turn on only when either both input 2 and 3 are true or If input 3 is true, while inputs 1 and 2 are false.*

Input 1	Input 2	Input 3	Active High Fan
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Example 2.10 shows that there are 3 states (or rows of the truth table) that provide true (or 1) outputs and since it is an active high device it will turn on when the output is true. Each of these rows are isolated so that all inputs are made to equal a true and then ANDed together so only the inputs that correspond with the row that is isolated will produce a true output from the AND gate. The following three bullets show how each of the true output rows are isolated.

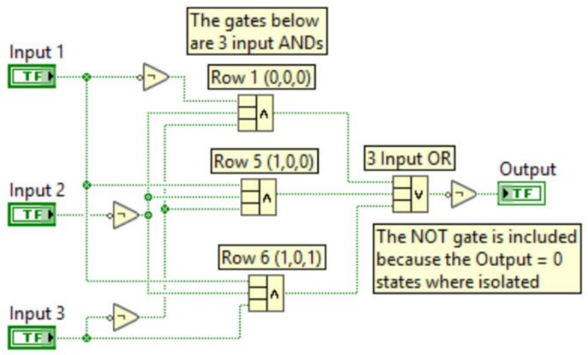
- Row 2 (Inputs 1 = 0, Inputs 2 = 0, Inputs 3 = 1) – Inputs 1 and 2 are flipped from a 0 to a 1 using NOT gates before they are wired (along with Input 3) to the 3-Input AND gate to produce a true AND gate output. The isolation occurs by only allowing a true to be the output of the 1st AND gate when the inputs are (0, 0, 1).
- Row 4 (Inputs 1 = 0, Inputs 2 = 1, Inputs 3 = 1) – Input 1 is flipped from a 0 to a 1 using a NOT gate before it is wired (along with Inputs 2 and 3) to the 3-Input AND gate to produce a true AND gate output. The isolation occurs by only allowing a true to be the output of the 2nd AND gate when the inputs are (0, 1, 1).
- Row 8 (Inputs 1 = 1, Inputs 2 = 1, Inputs 3 = 1) – All inputs are true so NOT gates aren't needed and each input is wired directly to the 3-Input AND gate to produce a true AND gate output. The isolation occurs by only allowing a true to be the output of the 3rd AND gate when the inputs are (1, 1, 1).

The final step of the Isolation-OR method is to OR together all of the outputs from the AND gates so that any of the isolated input combinations (i.e. "states" or rows of the truth table) can cause the output to be true.

If there are less false outputs than true outputs, then the zero outputs can be isolated and the final output of the OR gate can be inverted with a NOT gate. This situation is shown in Example 2.11.

Example 2.11) *Create the digital logic circuit using the Isolation-OR method from the truth table given below. Since there are more 1 outputs than 0 outputs isolate the 0 outputs and add a NOT gate after the OR gate.*

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1


The gates below are 3 input ANDs
Input 1
Row 1 (0,0,0)
Row 5 (1,0,0)
3 Input OR
Output
Input 2
The NOT gate is included because the Output = 0 states where isolated
Row 6 (1,0,1)
Input 3

The isolation-OR method frequently relies on multiple input gates, but these gates can be easily created from standard 2-input gates. Figure 2.14 shows the circuit in Example 2.11 implemented with only 2-input gates.

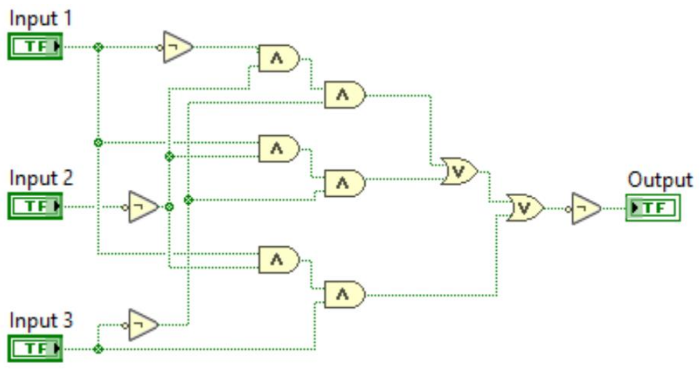


Figure 2.14: Logic circuit in Example 2.11 implemented with only 2-input gates.

Module 2 – References and Links

The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[4] More information about FETs and how they are used to create logic gates can be found at: https://www.allaboutcircuits.com/textbook/digital/chpt-3/cmos-gate-circuitry/

[5] Source for logic levels in Table 2.1: https://en.wikipedia.org/wiki/Logic_level

[6] Source for logic levels in Table 2.1: https://www.allaboutcircuits.com/textbook/digital/chpt-3/logic-signal-voltage-levels/

[7] More info on Boolean Algebra and gate text symbols: https://en.wikipedia.org/wiki/Boolean_algebra

[8] The CMOS circuit implementations of gates: https://www.allaboutcircuits.com/textbook/digital/chpt-3/cmos-gate-circuitry/

[9] Creating gates from only NAND and NOR gates: https://www.allaboutcircuits.com/textbook/digital/chpt-3/gate-universality/

[10] 7400 logic family part numbers: https://en.wikipedia.org/wiki/List_of_7400_series_integrated_circuits

[11] Digital eBook: https://www.allaboutcircuits.com/textbook/digital/

[12] Bit masking info: https://en.wikipedia.org/wiki/Mask_(computing)

[13] Computing checksums and common algorithms: https://en.wikipedia.org/wiki/Checksum

[14] More information about flags: https://en.wikipedia.org/wiki/Status_register

[15] SN74AHC132N Quad NAND gate datasheet: http://www.mouser.com/ProductDetail/Texas-Instruments/SN74AHC132N/?qs=sGAEpiMZZMuyBeSSR239IT5Ta765RsfIoGDW2r1dXUM%3d

[16] More information about Schmitt triggers: https://en.wikipedia.org/wiki/Schmitt_trigger

[17] Davis, *AC Circuits,* 2017, https://shareok.org/handle/11244/51946

[18] SLOA067 – *Op Amp and Comparators – Don't Confuse Them!:* *http://forums.parallax.com/discussion/download/96248&d=1350228174*

Module 3 – Measurement and Instrumentation Overview

Before discussing some of the sensors (see Module 4) that are commonly used in electromechanical systems, some background in measurements and instrumentation is needed.

Section 3.1 – Measurement Definitions

When determining how accurate a measurement is, the percent error (% Error) is usually used as the performance metric. The % Error is defined in Equation 3.1.

[3.1] **% Error = 100 · (Measured Value – True Value) / (True Value)**

If the **(measured value > true value) → (% Error > 0)** and if the **(measured value < true value) → (% Error < 0)**.

Generally, errors can be put into the following three categories.

1. **Random Errors** – For this type of error, the same measurement can be made over and over again, but different measured values are obtained due to a variety of different reasons, such as inconsistencies in the measurement process, instrument, or method. Random errors cannot be completely eliminated, but they usually have a normal distribution and can typically be reduced by taking the average of multiple measurements. If there were no other errors and the random error had a normal distribution, then theoretically the true value could be determined by taking the average value of infinite measurements.
2. **Bias Errors** – For this type of error, the measured value is consistently higher or lower than the true value. Bias errors are often called systematic errors. The most common type of bias error is due to a measuring device not being calibrated correctly so that the instrument itself has a bias in its measurement. Another example of bias error is related to the number of bits used to store the measured value. If the analog measured value is rounded to the nearest digital value (Analog to Digital conversion will be discussed later) then the reading will likely be rounded up or down, which will result in a bias error. Another example of bias error is related to the location of the measurement, which is often referred to as a spatial error. For example, if the temperature was measured in a tank, the fluid would likely have different temperatures at different locations in the tank. If the "true value" is considered the value in the center of the tank, there would be a spatial error as the location varied.
3. **Transient Errors** – This type of error occurs when the quantity being measured is fluctuating and has not reached a steady state condition. The % Error equation shouldn't be used in this case because the true value and the measured value are both changing. One common source of transient error occurs when the measuring instrument affects the measurement. For example, a cold temperature probe could cool down fluid in a tank and the true value prior to putting the probe in the tank would no longer be "true."

Example 3.1) *The true value of temperature is known to be 27.1 ^oC. If the system is at steady state and five measurements are made and the results were: 26.93 ^oC, 26.89 ^oC, 27.23 ^oC, 27.31 ^oC, and 27.05 ^oC. What is likely the primary type of error that is occurring? Estimate the measured value and the % Error from these readings.*

Since the system is at steady state and the readings are above and below the true value instead of consistently high or low, **random errors** are likely the primary cause of the error. To minimize the error the average of the 5 readings can be taken as (26.93 + 26.89 + 27.23 + 27.31 + 27.05)/5 = 27.082

Estimated measured value = **27.082 ^oC**

% Error = 100 · (Measured Value – True Value) / (True Value) = 100 · (27.082 - 27.1)/(27.1) = **-0.06642%**

Example 3.2) *The true value of temperature is known to be 27.1 ^oC. Several measurements are made and they are all very nearly equal to 27.5 ^oC. What is likely the primary type of error that is occurring? Find the % Error.*

Since the readings are consistently above the true value, **bias errors** are likely the primary cause of the error.

% Error = 100 · (Measured Value – True Value) / (True Value) = 100 · (27.5 - 27.1)/(27.1) = **+1.476015%**

✓**Basic measurement concepts** that are often confused involve the terms **accuracy** and **precision.**

❖ **Accuracy** (or how little of error there is) is described as how close the measured value is to the true value.

❖ **Precision** is defined as the repeatability of the measurement. Example 3.2 showed a situation where the measurement system has excellent precision because all of the measurements were close to the same value, but the accuracy is considerably higher than the situation in Example 3.1.

✓**Fundamental concepts** in measurement systems involve the terms **sensitivity**, **resolution, analog**, and **digital.**

❖ **Sensitivity** is the electrical output of the sensor divided by the property that is measured (e.g. $\mu V/^oC$ in a thermocouple). Some sensors, like thermocouples, have poor sensitivity because their electrical output is small for a given change in the property that is measured. A sensor with poor sensitivity might not be able to accurately measure a physical property because the noise floor (i.e. amount of noise present in the signal) has an amplitude comparable to the signal amplitude from the sensor. When a signal has poor sensitivity, it typically needs to have the amplitude increased with an amplifier. To avoid amplifying noise picked up in the line, it is ideal to amplify the signal close to the location that the measurement is taken. The signal might also need to have the noise filtered out so the signal from the sensor can be more easily distinguished. Amplification and filtering are two types of Signal Conditioning, discussed in detail in Section 3.4. If a signal has a linear relationship between the electrical output and the physical property input, then the sensitivity is constant and can be used as a direct conversion between the sensor's physical input (e.g. temperature, light, etc.) and the sensor's electrical output (e.g. voltage, current, etc.). Examples of sensors (covered in Module 4) that are approximated as linear are: thermocouples, IC temperature sensors, LVDTs, and piezoelectric sensors. For example, if a hypothetical pressure sensor has a linear pressure to current relationship and the slope of that linear relationship is 5 psi/mA then the following could be used to convert between input and output. **The process shown in the bullet below is used for all sensors that can be approximated as linear.**

Example: If k_T = sensitivity = 5 psi/mA & the output measures 3.5 mA, it corresponds to 5*3.5 = 17.5 psi

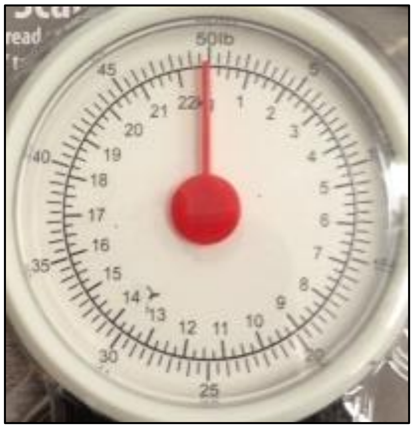
❖ **Resolution**, in general terms, refers to how small of a change can be detected when making a measurement.

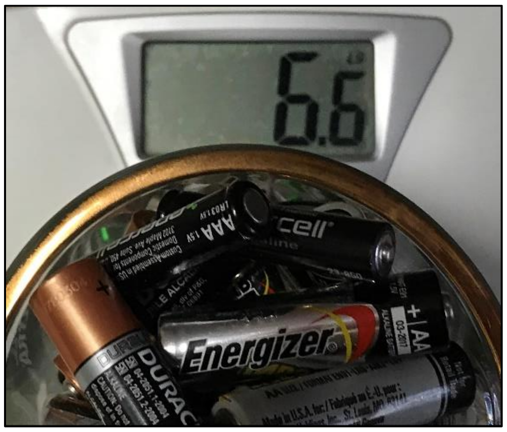
❖ **Analog**, in general terms, refer to any signal that has infinite resolution, which means if the physical property could be measured closely enough it would go out infinite decimal places. For instance, a sensor that has an analog voltage output will convert a physical property to a voltage according to some sort of mathematical relationship, but it won't limit the output to set voltage levels. For the analog measurement to be processed by a CPU it can be passed to an analog input in a data acquisition system (DAQ) as discussed in Section 3.2.

❖ **Digital**, in general terms, refer to any signal that has finite resolution and is limited by the number of bits used to store the measured value or the number of decimal places it is rounded to. For example, a sensor that has a "digital" output usually refers to a sensor that has an output that is either a high or low logic level (usually according to TTL logic levels, described in Table 2.1) so it can be passed directly into the digital input pin of a DAQ. In DAQ the digital input will be read into the computer as a Boolean (True or False).

The resolution of an analog instrument is simply the amount of change between each of the tick marks. For example, the analog weight gauge in Figure 3.1a has a new tick mark every 0.5 lb. The observer must decide visually which tick mark the red arrow is closest to and then record that value, so that the recorded value could be said to be limited to a 0.5 lb. resolution. In recording the digital number from the analog scale, the analog to digital (A to D) conversion is essentially being done with the observer's eyes. That visual approximation recorded by the observer could be considered a digital value because it has finite resolution.

Figure 3.1b shows a digital weight scale measuring a mug full of batteries (if you don't have a mug full of batteries you should get one!). The scale has only one decimal place on its display, but it can only read 0, 2, 4, 6, or 8 for the digit after the decimal point. Instead of relying on an observer to approximate the value from the arrow on the gauge, like in the analog gauge, the number is displayed on the read out in this digital gauge. The resolution of the digital weight scale below is 0.2 lbs. because that is the smallest change that can be detected.


(a)



(b)

Figure 3.1: (a) Analog weight scale with a 0.5 lb. resolution. (b) Digital weight scale with a 0.2 lb. resolution.

The digital weight scale is a great example of a complete measurement system because it contains a sensor, an A to D converter (ADC), and a **microcontroller**. An example of the type of load sensor used on digital weight scales is a SparkFun SEN-10245. The following link has a video that shows how to take apart a digital scale and use the internal load sensor to make your own digital scale. http://www.nerdkits.com/videos/weighscale/

Note: *A **microcontroller** is basically a small computer that is implemented on an IC. The brains of both the microcontroller and computer is the central processing unit (CPU). For convenience, the term CPU will often be used when referring to either a microcontroller or a computer in this book.*

Analog to Digital Converter (ADC)

For computerized measurement applications, the resolution is limited by the ADC process. ADCs (or **digitizers**) are designed to store each number in a fixed number of bits so that it converts an analog (infinite resolution) signal to a digital (finite resolution) signal. The number of bits (N) and the range of the measurement are used to calculate the ADC resolution as shown in Equation 3.2.

[3.2] **ADC Resolution = Range / (2^N)** where: Range = max value – min value & N = Number of bits used

Note: *The term "resolution" is used in many different ways and in many different applications. For instance, Wikipedia has dozens of different pages for different usages of the term "resolution." In terms of measurement and instrumentation theory, the basic definition of resolution is the smallest change that can be detected and this definition is put in equation form in Equation 3.2 and applies to the ADC process. From this point on in the book, ADC Resolution will simply be called resolution.*

In Equation 3.2, the number of bits (N) is used to determine the total number of steps (or bit combinations) for the digital value that is stored in the computer. For example, if you stored a number in 3 bits you would only have 8 (or 2^3) different possible bit combinations for the number, which would likely not be enough. Example 3.3 shows the resolution calculation for different numbers of bits. This example shows that in order to reduce the resolution the number of bits needs to be increased. It is not wise to increase the number of bits to a number that is larger than needed because it results in more memory being required to store the numbers and requires more expensive A to D hardware. The goal is usually to determine the **minimum number of bits** that can be used to achieve a **resolution that is no greater than** the desired limit for a situation (see Example 3.3).

This book is titled **Electromechanical Systems** because the main topics that go into making an electromechanical system are covered in the book. The following figure shows an example of an electromechanical system. It has an embedded processing unit (NXT Brick) that has a microcontroller used to control the 4 sensor input ports (labeled 1-4) and the 3 actuator (NXT motors) output ports (labeled A, B, and C). It includes a single ADC to read the sensor values in a multiplexing (or scan mode) process (multiplexing is discussed in the next section). Three external sensors are plugged into ports 1, 2, and 3, but the NXT motors also include internal optical encoders (a cut away of an NXT motor showing the encoder is shown in Figure 4.25) that are also sampled with the ADC. Detailed specifications about the Lego Mindstorm NXT and a comparison with the newer Lego EV3 is shown at the following link: http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/

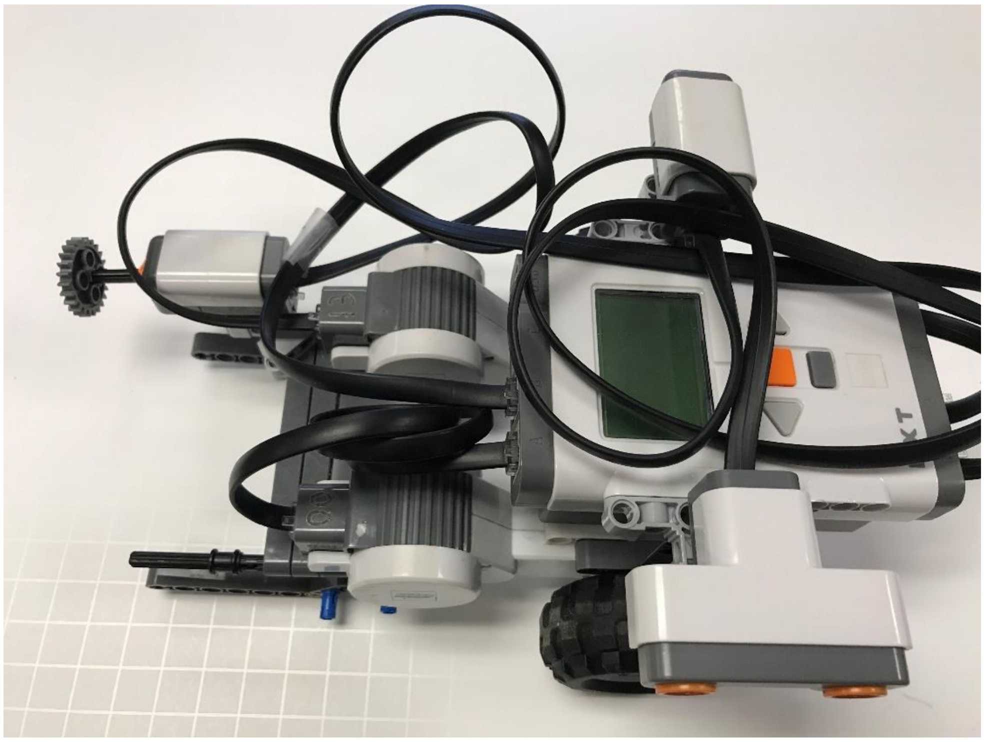


Figure 3.2: Lego Mindstorm NXT robot with Ultrasound ranging sensor (aiming down), a light sensor in ambient mode (aiming up), and a push button (or pressure) sensor facing left.

Example 3.3) *Using a current range from 4 to 20 mA (which is a common standard in measurement and control systems and described in this link: https://en.wikipedia.org/wiki/Current_loop), calculate the resolution when using four different A to D converters with the following numbers of bits: a) 3, b) 12, c) 16, d) 24.*

The range is equal to the max – min → Range = 20 – 4 = **16 mA**

The units of resolution is the units of the range divided by "steps," where there are 2^N number of steps.

a) N = 3 → Resolution = Range / (2^N) = 16mA/2^3) = **2 mA/step**
b) N = 12 → Resolution = Range / (2^N) = 16mA/2^{12}) = 0.003906 mA/step or **3.906 µA/step**
c) N = 16 → Resolution = Range / (2^N) = 16mA/2^{16}) = 0.000244 mA/step or **0.244 µA/step**
d) N = 24 → Resolution = Range / (2^N) = 16mA/2^{24}) = 9.54E-7 mA/step or **0.954 nA/step**

This example shows that if a resolution no greater than 1 nA/step is desired for a 4 to 20 mA measurement range, a 24 bit A to D converter is a good choice because it meets the specifications, but doesn't include extra bits that results in wasted memory and increased cost of the system. Likewise, if a resolution no greater than 4 µA/step is desired a 12 bit A to D converter is the ideal choice. Even though the 16 and 24 bit converters would also have sufficient resolution (i.e. less than 4 µA/step), they have extra bits that are not needed and wasteful.

For example, If you needed "no greater than" 1.6 µA/step, which (a, b, c, or d) is the best option? **16 bit ADC**

Going back to the digital scale example, discussed previously, the resolution from the ADC usually is a smaller number than displayed to the user (or observer) in a system with a digital read out because numbers are usually represented to humans as decimal or fractional (base 10) instead of binary (base 2) or hex (base 16). For the digital scale, the internal ADC resolution must be "no greater than" 0.2 lb/step in order to provide the observer a 0.2 lb/step display resolution on the screen. The next example shows that the finite resolution resulting from the ADC process produces a bias error.

Example 3.4) *If a 3 bit ADC is used in a measurement system that has a range of 0 to 5V, correlate each of the binary numbers that make up the 8 steps to a voltage level. If a sensor with a near 100% accuracy measured a value of 2.36 V with an ADC that had an infinite number of bits and there were no other errors in the measurement system, what bias error results from the 3 bit ADC.*

N = 3 → Resolution = Range / (2^N) = 5/2^3) = **0.625V/step**

The 8 states resulting from using 3 bits could be set as follows:

- 000 → 0 V
- 001 → 0.625 V
- 010 → 1.25 V
- 011 → 1.875 V
- 100 → 2.5 V
- 101 → 3.125 V
- 110 → 3.75 V
- 111 → 4.375 V

In this hypothetical example with perfect accuracy and no errors, the measurement of 2.36 V is considered the True Value. When using the 3 bit ADC, 2.36 V would be mapped to the closest binary number of 2.5 V (from the binary 100 state). The resulting bias error is 2.5 – 2.36 = 0.14 V and the % Error could be calculated as:

% Error = 100 · (Measured Value – True Value) / (True Value) = 100 · (2.5 – 2.36)/(2.36) = **5.9322 %**

✓**Other more advanced concepts** in measurement systems involve the terms **sampling**, **Nyquist**, and **aliasing**. References [8], [9], and [10] contain more information about these topics.

❖ **Sampling** is a term used to describe the process of using the ADC to acquire a measurement. The sampling rate (or frequency), F_S, is the number of samples per second that can be acquired. The units of kHz (kilo cycles per second) or kS/s (kilo samples per second) are usually used for sampling rate in most DAQ devices. Table 3.1 shows examples of DAQ units and includes a comparison of the sampling rate.

❖ **Nyquist** refers to the minimum sampling frequency (**F_{min}**) that can be used in order to reconstruct the signal. **$F_{min} = F_{max}/2$**, where **F_{max}** is the maximum frequency being measured. For example, Compact Disc (CD) audio is sampled at a rate of 44.1 kHz. This rate was specified for CD audio because it is slightly over twice the human audio range, which is approximately 20 kHz. So, if F_{max} = 20 kHz, the Nyquist rate (**F_{min}**) = 40 kHz.

❖ **Aliasing** occurs if a signal is sampled at less than the Nyquist rate (**F_{min}**). Aliasing results in a set of sampled data points that, when reconstructed, produces a signal at a different frequency than the original (as shown in Figure 3.3). For example, if the maximum frequency of a signal was thought to be 1 kHz, then F_S could be appropriately selected to be 2 kHz because it is 2*F_{max} (**Note:** *Ideally, F_S would be set a higher than 2*F_{max} as explained in the CD audio example*). However, if there was some unexpected high frequency noise in the measurement that was higher than 1 kHz (i.e. > 0.5*F_S) it would result in a frequency component that is "aliased" into the 0 to 1 kHz range. The aliased signal cannot be distinguished from the actual signal (that was thought to be < 1 kHz) and thus cannot be filtered out in software after the measurement is made. Since unexpected frequencies that are higher than 0.5 times the sampling rate (F_S) frequently occur, "anti-aliasing" low pass filters are often used prior to sampling a signal to assure no frequencies are present beyond 0.5*F_S. Figure 3.3 shows cosine and a sine signals (gold) that are sampled (black dots) and higher frequency aliases (blue) of these signals. Notice, the aliases travel through the same sampled points as the original signals. If the blue signal was sampled at a F_S below the Nyquist rate (**F_{min}**) then it would be reconstructed as a signal with a lower frequency, as demonstrated in Figure 3.3.

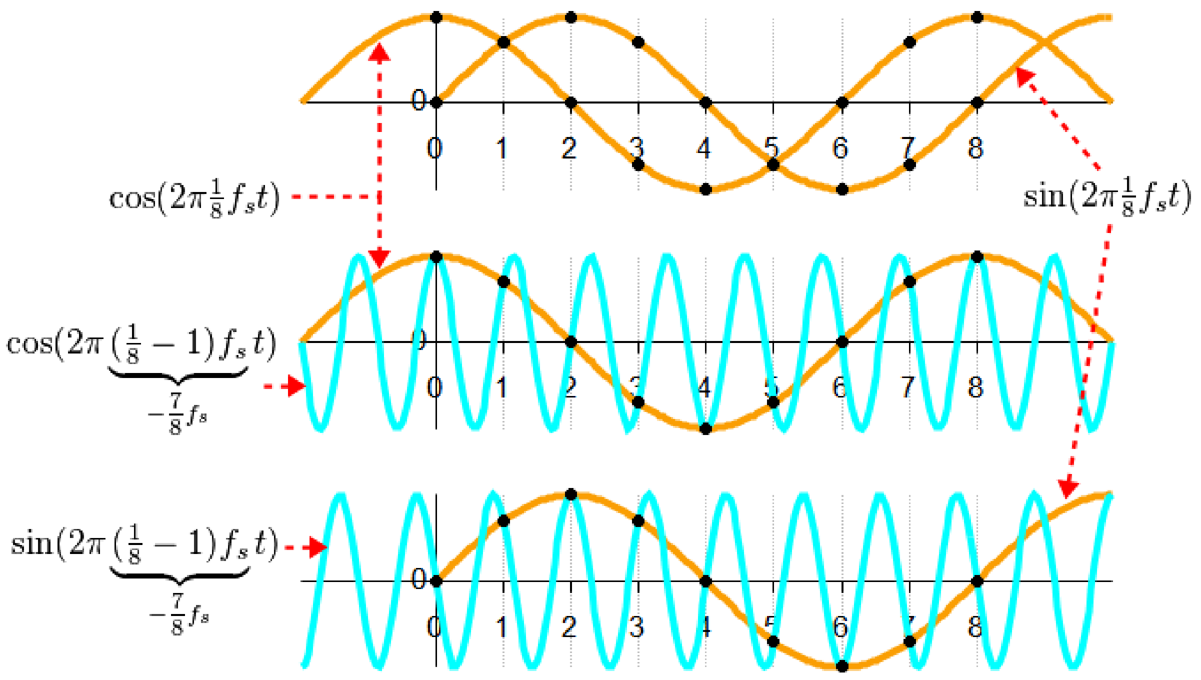
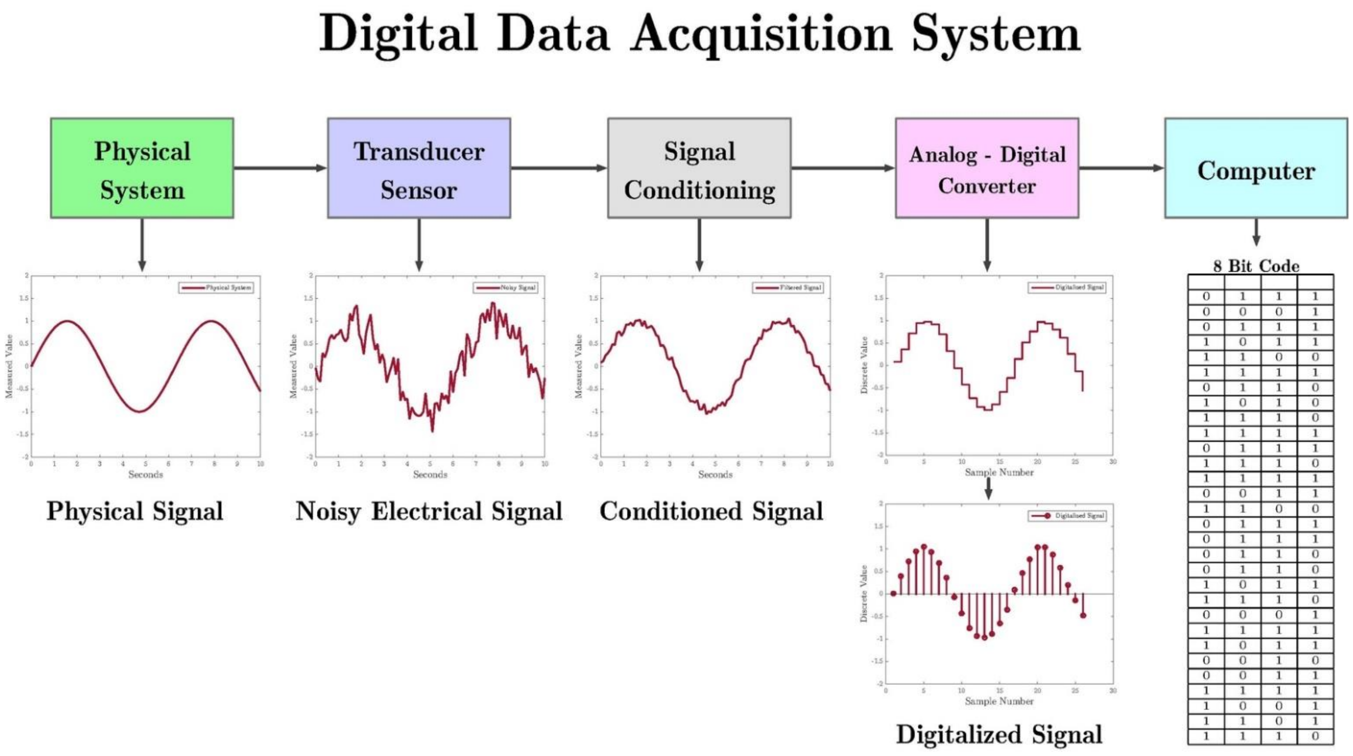


Figure 3.3: Illustration demonstrating aliasing, (Gold = original signals, Blue = aliased signals). Public Domain: https://en.wikipedia.org/wiki/Aliasing#/media/File:Aliasing_between_a_positive_and_a_negative_frequency.png

Section 3.2 – Data Acquisition (DAQ) Overview

Data acquisition (DAQ) allows a physical quantity to be measured and the data transferred to a computer. Figure 3.4 shows a block diagram that includes the basic elements of a DAQ system.



0	1	1	1
0	0	0	1
0	1	1	1
1	0	1	1
1	1	0	0
1	1	1	1
0	1	1	0
1	0	1	0
1	1	1	0
1	1	1	1
0	1	1	1
1	1	1	0
1	1	1	1
0	0	1	1
1	1	0	0
0	1	1	1
0	1	1	1
0	1	1	0
0	1	1	0
1	0	1	1
1	1	1	0
0	0	0	1
1	1	1	1
1	0	1	1
0	0	1	0
0	0	1	1
1	1	1	1
1	0	0	1
1	1	0	1
1	1	1	0

Figure 3.4: Data Acquisition System Block Diagram. © Enrique Z.L. Used under CC BY-SA license: https://en.wikipedia.org/wiki/File:DigitalDAQv2.pdf

The first two blocks in Figure 3.2c show that the first two steps of a DAQ system is a sensor (or transducer) that is used to make measurements from a physical system. The electrical signal being measured could range anywhere from a complicated waveform with multiple data points to a single scalar value. Different types of sensors that are used to measure various types of physical quantities are covered in detail in Module 4, but this section will focus on the last three blocks of the DAQ process. Different types of signal conditioning are covered in Section 3.4.

DAQ can be performed in many ways, but this book will focus primarily on using LabVIEW to control a National Instruments (NI) DAQ device. There are numerous links in the Module 3 – Reference and Links section that provide additional details of DAQ beyond what is covered in this section. The following document from National Instruments (NI) provides a good DAQ overview: http://www.ni.com/data-acquisition/what-is/

The most convenient and easy way to build a DAQ system is to use a NI-USB DAQ device that connects to a computer with a USB cable. Using NI Daqmx device drivers and the LabVIEW programming environment, these devices allow measurements to be made easily. A NI-USB device that is marketed to academia is called the myDAQ. The myDAQ works essentially the same way as other NI-USB devices, but it also has a built-in Multimeter and audio in and out ports. Table 3.1 shows specifications of a variety of NI-USB DAQ devices (including the cheapest and most expensive USB DAQ devices offered by NI at the time of publication). A full list of NI DAQ devices can be found here: http://www.ni.com/en-us/shop/select/multifunction-io-device

Table 3.1) Specifications for Various NI-USB DAQ devices (D = Differential, SE = Single Ended)

Specification	myDAQ	USB-6366	USB-6211	USB-6000
# of Analog Inputs	2D	8D	8D or 16SE	8 (SE only)
Input Voltage Range (±)	2 or 10 V	1, 2, 5, or 10 V	± 0.2,1,5, or 10V	± 10 V only
# of Analog Outputs	2	2	2	0
Output Voltage Range (±)	2 or 10 V	5, 10 V, or external ref.	10 V	N/A
Max Sampling Rate	200 kS/S	2,000 kS/S	250 kS/S	10 kS/S
Min Sampling Period	5 µs	0.5 µs	4 µs	100 µs
# of Digital Inputs/Outputs	8 (In or Out)	24 (In or Out)	4 In, 4 Out	4 (In or Out)
# Counter Inputs/Outputs	1	4	2	1
ADC Resolution (# of bits)	16	16	16	12
Sampling Type	MUX	Multiple-ADC	MUX	MUX
Cost as of 2018	$358	$4,884	$973	$165

The "Sampling Type" specification for three of the devices in Table 3.1 is listed as MUX, which is an abbreviation for multiplexing. With low cost DAQ devices, the analog to digital converter (ADC) internal hardware is shared between the channels and only one measurement is performed at a time and done in a sequential manner. When multiple measurements are made sequentially it is called multiplexing. **Multiplexing** with a DAQ is also referred to as **scan mode** because the measurements are made by scanning through each of the inputs (that are set up to be taken) one at a time. More expensive DAQ devices with multiple ADCs (such as the X-series USB-6366 in Table 3.1) have the capability to take multiple measurements simultaneously, so that the data points are taken at almost exactly the same time. To avoid the expense of separate ADCs on each channel, there other methods that attempt to align data that is taken at different times with a MUX device. There is a hardware method called SSH (**S**imultaneous **S**ample and **H**old) and various software methods that are used to try to align the data. These hardware and software synchronization methods, as well as systems with true simultaneous sampling capabilities that include multiple ADCs, are described in the following link: http://www.ni.com/white-paper/4105/en/

Note: *Most of the multiple ADC DAQ devices have PCI, PXI, or PXI express interfaces instead of USB.*

Most DAQs have digital pins that can be configured as inputs or outputs (DIO). Example 3.7 shows how to use the DAQ Express VIs to configure digital inputs and outputs. Each pin is called a "line" and **8 lines make up a port**. The DAQ can be configured to read from or write to one line at a time or an entire port at a time. The far right DAQ Express VI in Example 3.7 shows that the 8-bit decimal equivalent value is used to write out to a port. When a Boolean true is sent to a digital output line from LabVIEW the signal on that pin is set to 5 V. The amount of current that is sourced from a digital output line increases as the resistance that it is connected to it is reduced, but the max current is limited internally in the DAQ (e.g. maximum of 4 mA for the myDAQ). When a DIO line is configured as a digital input, TTL logic levels (as described in Table 2.1) are usually used. When a digital line detects a logic high, a Boolean true will result in LabVIEW.

Counters are special digital pins that allow for digital pulses to be created or measured. The counter pin is often one of the DIO pins that can be configured as either a standard digital line or as a counter. For the myDAQ, only DIO3 can used as a counter and this link shows how to use the counter pin. When a counter is configured as an input, it can be used to "count" the number of pulses received or to measure parameters of the digital signal, such as **D**uty **C**ycle (**DC** = $t_{high}/(t_{low} + t_{high})$ or frequency (F = $1/(t_{low} + t_{high})$). When a counter is configured as an output, a signal that is a 5V pulse for a prescribed amount of time can be sent out or a periodic square wave with prescribed high time (t_{high}) and low time (t_{low}) can be sent out.

Analog inputs can be configured to receive one sample at a time or receive blocks of data (N samples). When N samples are read on the analog input line, the sampling rate (F_S) can be set to any frequency up to the max sampling rate limit (The F_S limit is shown for some DAQs in Table 3.1). Analog outputs can be configured to send out a single data point (any voltage in the DAQ voltage range) or multiple data points that make up a waveform. When a waveform is sent out of the analog output pin, it is usually given a specific amplitude and frequency and is usually either a periodic sine wave, square wave, or triangle wave. Example 3.9 shows how to use the DAQ Express VI to receive N samples into an analog input pin from an electret microphone and write a sinusoid waveform out of the analog output. The "Simulate Signal" Express VI is used in that example to create a sinusoid at the same frequency that is measured from the electret microphone signal.

Note: *The sampling frequency is often called sampling rate. The units are listed as kS/s (or kilo samples per second), which is equivalent to kHz. The sampling period is equal to the inverse of the sampling frequency and defines how often a new sample can be acquired. For example, if N=100 and the sampling rate is set to 200 Hz it would take N/F_S = 0.5 seconds to acquire all 100 samples.*

A summary of the different types of inputs and outputs in a DAQ are shown below.

- Digital inputs – Can receive a single data point using TTL logic levels to turn on or off Boolean indicators.
- Counter inputs – Can be used to "count" the number of pulses received or to measure parameters in a digital signal, such as frequency or duty cycle.
- Analog Inputs – Can receive one sample at a time or receive blocks of data (N samples). When N samples are read on the analog input line, each sample can be read at a speed up to the max sampling rate limit. When multiple analog inputs are measured, multiplexing (MUX) is usually used and the max sampling rate is reduced by the number of inputs being measured.
- Digital outputs – Can only send a single data point out on each line at 0 or 5V.
- Counter outputs - Can send out a 5V pulse for a set time or a 0 to 5V adjustable duty cycle square wave.
- Analog Outputs – Can send out a single data point or a waveform (i.e. sine wave, square wave, triangle wave) at a set frequency and duty cycle at any voltage in the DAQ voltage range.

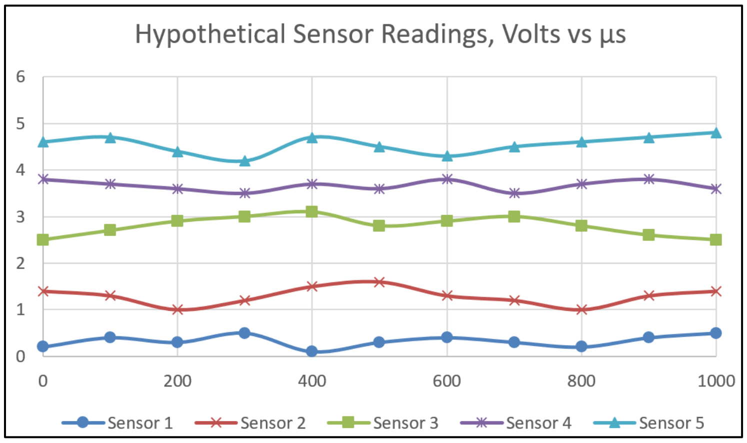
Appendix A, provides a detailed step by step guide on how to use LabVIEW with a myDaq to take measurements of a sensor and includes all 6 types of DAQ inputs and outputs (analog, digital, counter).

Example 3.5) *What is the shortest amount of time it would take to measure 100 samples from one analog input and 60 samples from a second analog input port using the myDAQ in N sample block processing mode?*

The following steps are needed to solve this problem.

- Table 3.1 states the maximum sampling rate for a myDAQ is **200 kHz**.
- This results in a minimum of **5 µs** (1/200,000) between samples.
- It would take **500 µs** (100 · 5 µs) to receive the first data packet of 100 samples.
- Since the myDAQ operates using multiplexing (or reads data in scan mode) the 60 samples from the 2nd analog channel will be received after the completion of reading the 100 samples from the 1st analog channel. To read the 60 samples it will take an additional **300 µs** (60 · 5 µs).
- Assuming there is minimal delay between the readings on the two analog channels, the total amount of time would be **800 µs** (500 µs + 300 µs)
- The 1st and 2nd analog inputs, that were mentioned in this example, can be set to any available ports (e.g. AI0 and AI1 in the myDaq) and they can be set to read the data in any order (e.g. For the myDaq either AI0 first, then AI1 or AI1 first, then AI0).

Example 3.6) *In general, if you used multiplexing to read 5 sensors with a data acquisition device that had a sampling frequency of 50 kS/s (kilo samples per second), what is the fastest update time (Sampling Period) you could achieve for each sensor?*



Since multiplexing is used, each sensor is read sequentially so it takes 5 times longer to read 5 sensors than 1 sensor. The fastest update time for each **sample** is equal to **20 µs** (1/50,000). Due to multiplexing with 5 sensors, the fastest update time for each **sensor** is equal to **100 µs** (5·20 µs), assuming there is minimal delay when switching between sensors. For example, the plot to the right shows an example of how the data from the sensors might look after the readings are aligned to have the same x-axis (**Note:** *Since multiplexing is used to make the measurements, they can't be measured simultaneously*). Notice that there is **100 µs** between each sensor data point instead of the 20 µs sampling period that would have been achieved if only one sensor was measured.

Four LabVIEW DAQ examples are shown in Section 3.3, where sensors and/or motors are connected to a NI myDAQ device. Specification for the NI myDAQ, and three other DAQ devices, where shown in Table 3.1. The DAQ examples in Section 3.3 make use of "Express VIs" in LabVIEW. These Virtual Instruments (VIs) allow you to double click on them to access a Graphical User Interface (GUI) that allows for easy configuration. The Express VI palette is shown in Figure 3.5. This palette can be called by right clicking on the Block Diagram screen in LabVIEW. There are multiple VIs (also referred to as blocks) under each of the 6 Express VI sub-categories shown in in Figure 3.5. Express VIs are good for beginners in LabVIEW due to their ease of use, but more "programmatic control" (i.e. changing parameters while the program is running instead of going through the GUI to manually change them) is available by building LabVIEW programs with non-Express VIs. An example of this additional functionality is shown with LabVIEW servo motor control (Example 3.10), where the lower level Daqmx VIs are used to allow more control of the motor position than capable with the DAQ Express VIs.

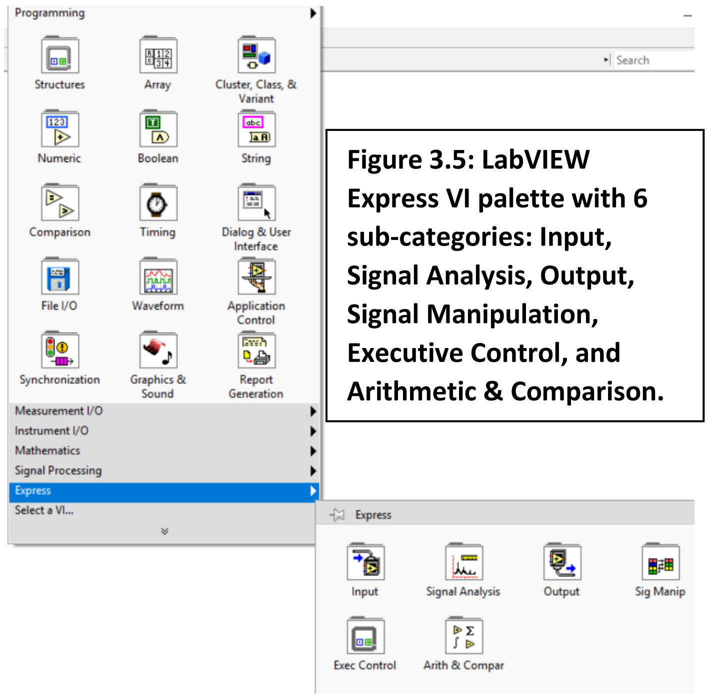


Figure 3.5: LabVIEW Express VI palette with 6 sub-categories: Input, Signal Analysis, Output, Signal Manipulation, Executive Control, and Arithmetic & Comparison.

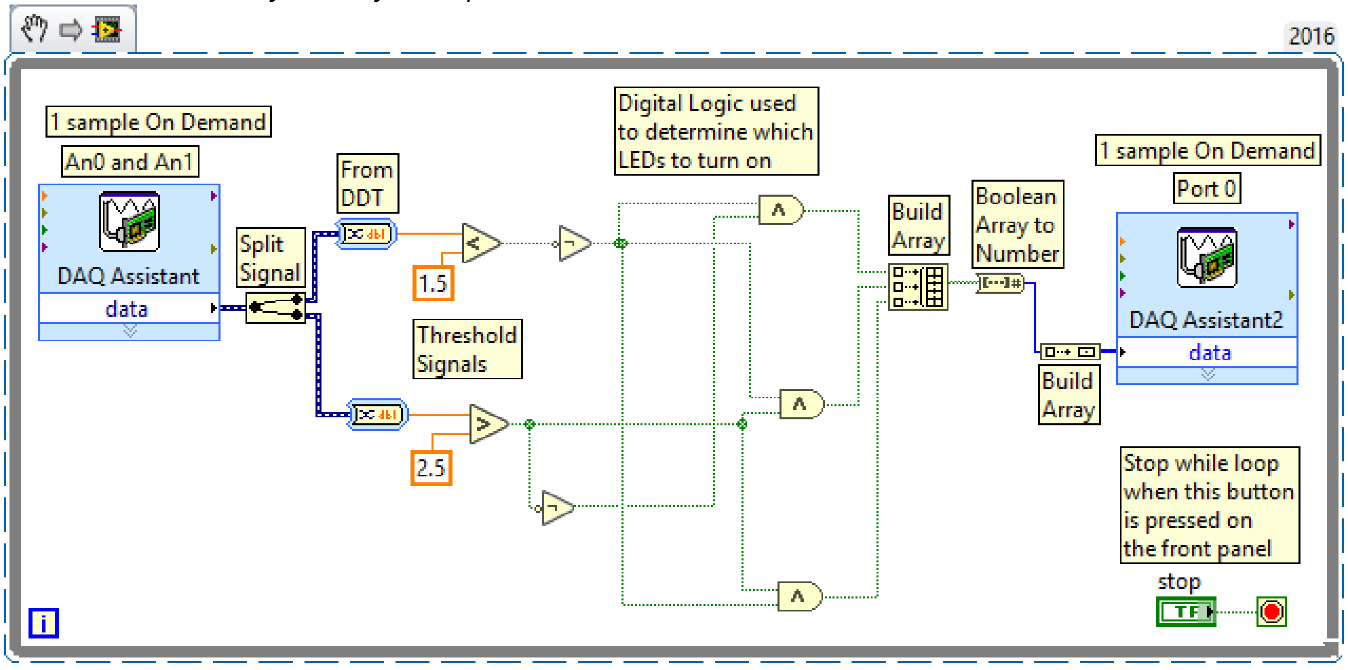
The Express VI palette is one of many different palettes available in LabVIEW. By default, the "Programming" palette is expanded to show all its sub-categories. Like the Express VI and the Programming palettes, the other palettes also have many sub-categories with many VIs under each one. There are also other palettes that can be selected by clicking on the two arrows directly below the "Select a VI" option. If navigating the palettes is too time consuming, VIs can also be found using the search feature in top right field in the Block Diagram or by holding the control key and pressing the space bar.

Section 3.3 – LabVIEW DAQ Examples

The following are example programs that cover the main aspects of using a DAQ. Both analog and digital signals from sensors and switches are measured and both analog and digital signals are sent out from the DAQ to control various devices such as Light Emitting Diodes (LEDs), speakers, and motors.

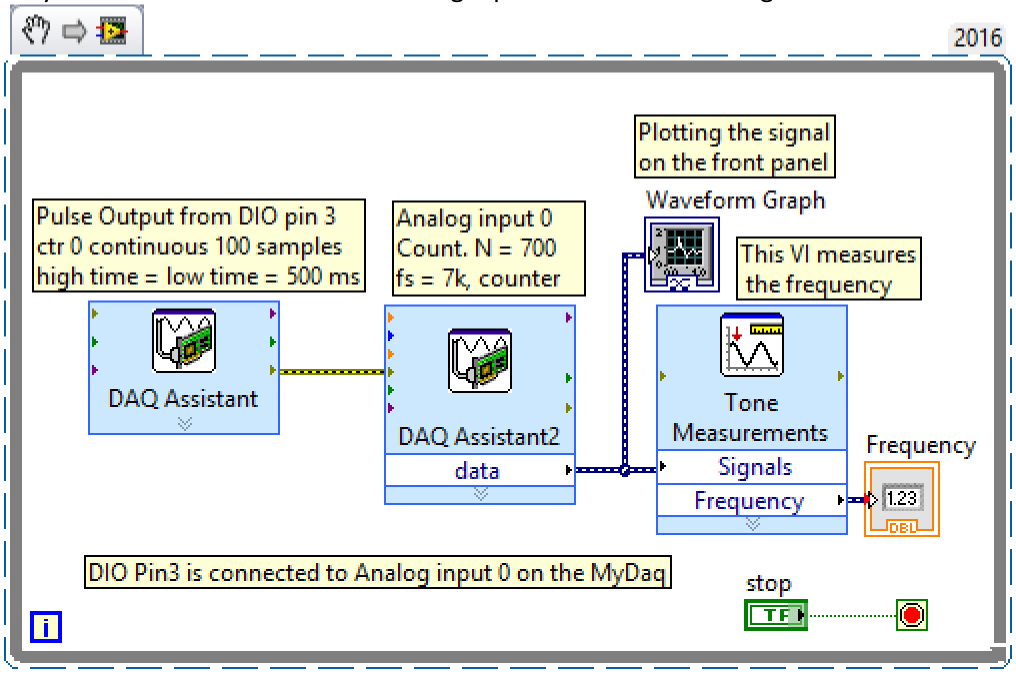
Example 3.7) *Use a Cadmium Sulfide Cell (photo resistor) to detect the ambient light level. Also, use an IR LED pointed at a phototransistor to detect whether there is an obstruction. You are to send these two sensor outputs to analog input channels on the DAQ. Set thresholds on the readings in LabVIEW to convert the signals to Boolean (True or False). Send the following Digital Output signals.*

- *When the light level is low, turn on the LED connected to channel 0.*
- *When there is an obstruction, turn on the LED connected to channel 1.*
- *When there is an obstruction AND the light level is low, turn on the LED connected to channel 2. Use the LabVIEW AND function for this part.*



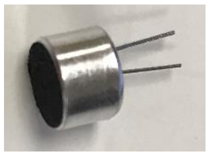
The LabVIEW code above shows that the DAQ assistant block is set so that analog input 0 (AI0) and analog Input 1 (AI1) are both used to measure the sensor signals using scan mode (multiplexing) using the "one sample on demand" scheme. Since the photo resistor has an output of resistance, a signal conditioning circuit must be used to convert resistance to voltage. This circuit is a simple voltage divider with a fixed resistor and a photo resistor (CdS cell) in series with the 5 Volt output from the myDAQ. The voltage can be measured across either the resistor or photo resistor to get a changing voltage as the light level changes. The IR LED and phototransistor are connected in the active low phototransistor configuration described in Section 4.2.2.
Since there is no "wait" command to slow down the while loop (the outside loop), the while loop runs as fast as possible. The myDAQ will take a measurement of AI0 first and AI1 next in scan mode (multiplexing). The data from this DAQ assistant block is sent out through a "split signal" block, which splits the AI0 and AI1 signals into two separate data streams. The data is then converted to a fractional number using the "Convert from Dynamic Data Type (DDT)" block. The sensor data is then thresholded and digital logic is used to set the LEDs to turn ON. In order to get the signal in the proper format for the DAQ to write to multiple pins on a port, two build array blocks with an "array to number" block separating them are used. Finally, using the DAQ Express VI, the myDAQ will write out to the digital port and the process will immediately start again in the next while loop iteration.

Example 3.8) *Use counters to send a pulse train to flash an LED at 1Hz. Measure the frequency of the blink rate with the DAQ.* The figure below shows how to create the LabVIEW code and explains how to wire the myDAQ. The counter output (DIO Pin 3) is wired to the Analog input 0+ in the MyDAQ so that the counter pulse train can be analyzed inside of LabVIEW. The Analog input 0- is connected to ground.

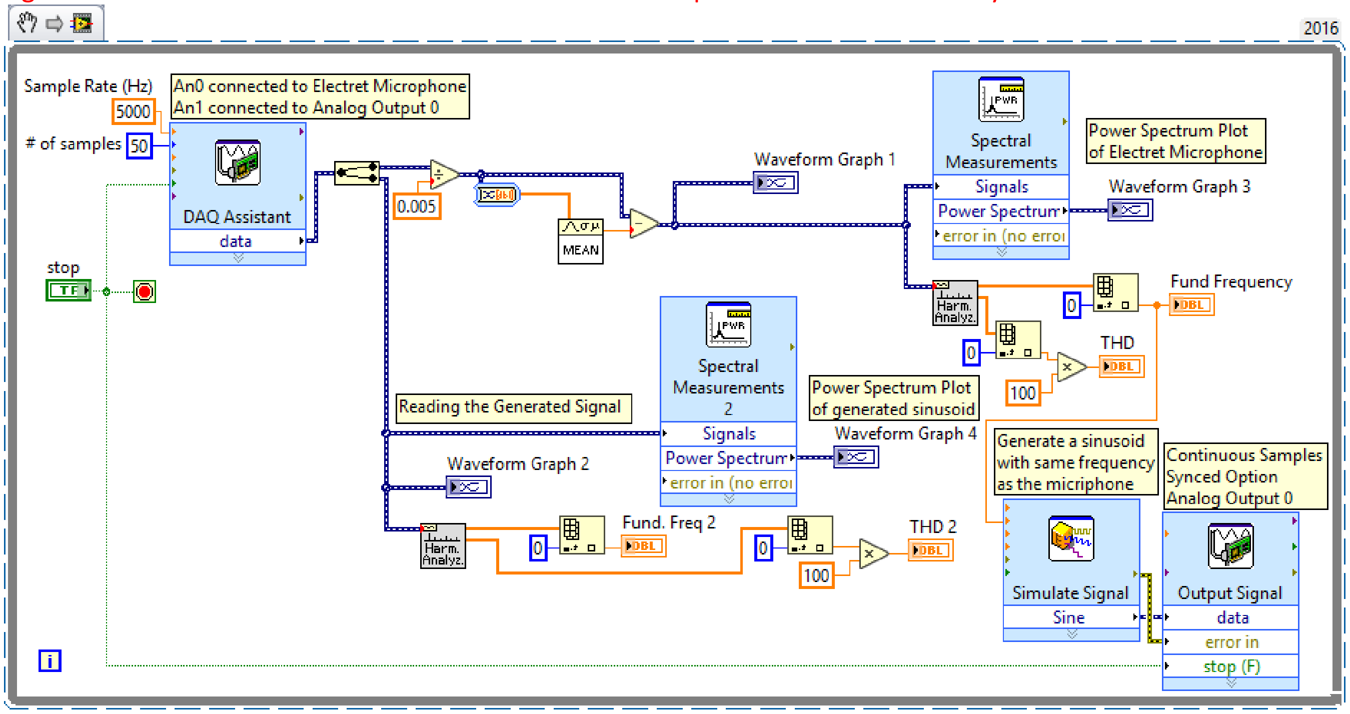


2016
Pulse Output from DIO pin 3
ctr 0 continuous 100 samples
high time = low time = 500 ms
DAQ Assistant
Analog input 0
Count. N = 700
fs = 7k, counter
DAQ Assistant2
data
Plotting the signal on the front panel
Waveform Graph
This VI measures the frequency
Tone Measurements
Signals
Frequency
Frequency
1.23
DBL
DIO Pin3 is connected to Analog input 0 on the MyDaq
stop
TF
i

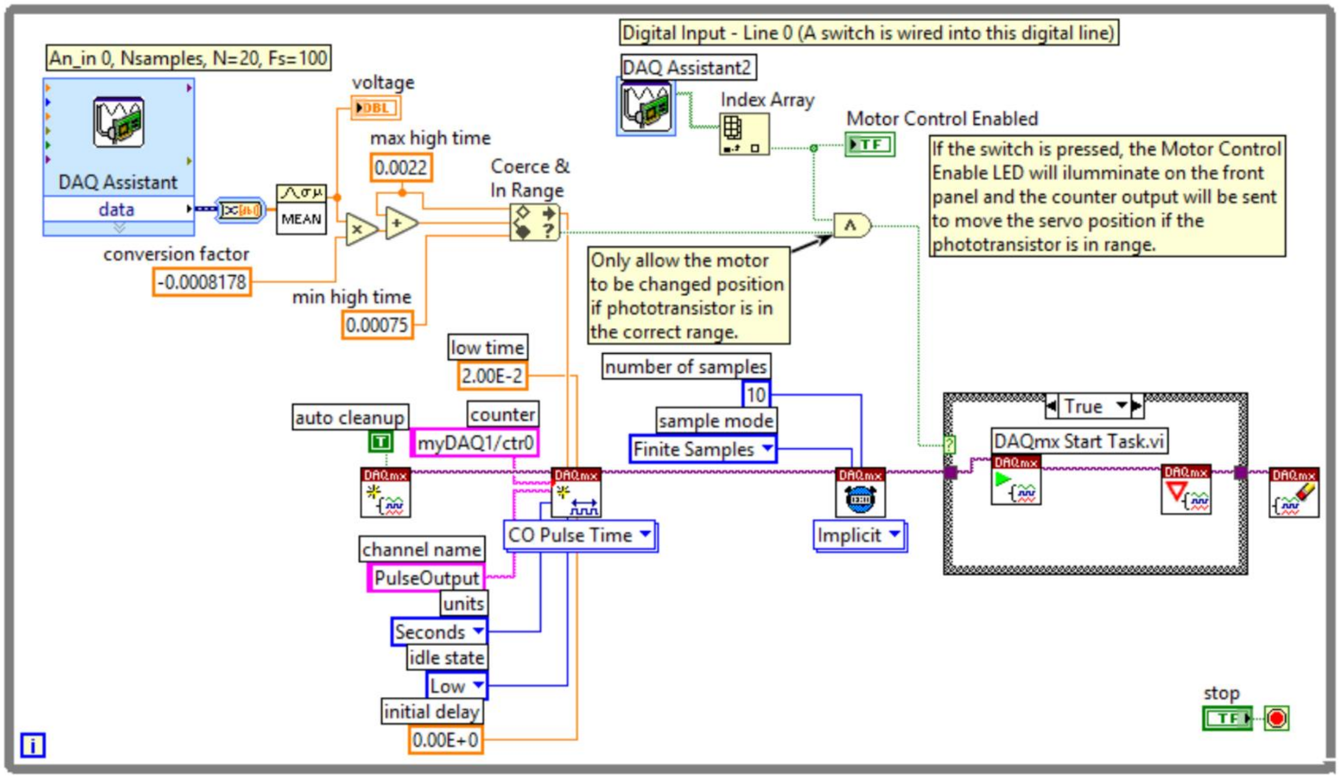
Example 3.9) *Whistle into an* electret microphone *(see photo to the right) & capture the sine wave in LabVIEW. Scale the sine wave in LabVIEW so that its peak amplitude is approximately 1V. Additionally, output a sine wave from the DAQ that has a similar frequency and amplitude as your other input and feed it into a second analog input channel. Measure the fundamental frequency and the total harmonic distortion (THD) for both of these sinusoids.*



The figure below shows the LabVIEW code for this task and explains how to wire the myDAQ.



Example 3.10) *Control a 180° [Servo Motor] with LabVIEW (See Spec. sheet for the [Parallax #900-00005]). Use an IR LED and phototransistor to make a range detection circuit that changes the position of the servo. When the IR LED is far away from the phototransistor, rotate the servo all the way counter-clockwise and when the IR LED is close to the phototransistor move the servo all the way clockwise. There should be a gradual transition on the servo rotation between the two extremes so that it is close to the center servo location at the midpoint of the far and close distances. Only allow the motor to move when a momentary switch is pressed and have a LED on the LabVIEW front panel called "motor control enabled" that turns green when the switch is pressed.*



An_in 0, Nsamples, N=20, Fs=100
DAQ Assistant
data
conversion factor
-0.0008178
voltage
DBL
MEAN
max high time
0.0022
min high time
0.00075
Coerce & In Range
Digital Input - Line 0 (A switch is wired into this digital line)
DAQ Assistant2
Index Array
Motor Control Enabled
TF
If the switch is pressed, the Motor Control Enable LED will iluminate on the front panel and the counter output will be sent to move the servo position if the phototransistor is in range.
Only allow the motor to be changed position if phototransistor is in the correct range.
low time
2.00E-2
number of samples
10
sample mode
Finite Samples
auto cleanup
counter
myDAQ1/ctr0
CO Pulse Time
Implicit
True
DAQmx Start Task.vi
channel name
PulseOutput
units
Seconds
idle state
Low
initial delay
0.00E+0
stop

To control this 180° servo, the high time of the pulsed signal from the counter (DIO 3 pin) must range between 0.75 ms and 2.2 ms and the low time must be approximately 20 ms for the servo to spin from one extreme to the other (these details are described in the data sheet for the servo motor). In order to programmatically control the high time precisely (and the location of the motor), the express counter VI (the first block shown in [example 3.8]) cannot be used. Instead, the Express VI can be converted to Daqmx code by right clicking on the express VI and selecting "Convert to Daqmx code". Once the lower level Daqmx code is created the high time can be programmatically controlled by scaling the voltage from the phototransistor to a range of 0.75 to 2.2 ms.

Note: *To save block diagram programming space, the "DAQ Assistant 2" block in the figure above that performs a digital input operation is displayed as an "icon" instead of the large express VI form factor that is the default. To change to the Icon display, right click on the block and select "View as Icon."*

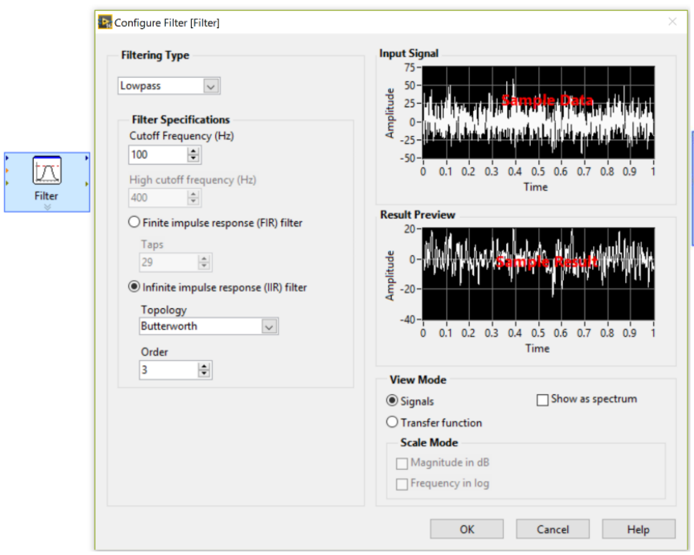
Section 3.4 – Signal Conditioning

Note: *You might want to click on the Figure 3.4 bookmark and press* ***Alt ←*** *to return back to this spot after reviewing the DAQ process again before reading this section.*

Figure 3.4 shows that signal conditioning is the next step in the DAQ process after the sensor converts a physical quantity to an electrical signal. Signal conditioning is not always necessary and when it is needed it can occur before or after the Analog to Digital Converter (ADC) and can occur outside the computer using hardware or inside the computer using software. Some common types of signal conditioning are discussed below:

- Filtering – An example of this type of signal conditioning is shown in Figure 3.4, where a noisy signal is shown below the block labeled "sensor" and most of the noise is removed in the signal shown below the block labeled "signal conditioning." Filtering is often necessary because noise is picked up by the sensor that can cause difficulties in making an accurate measurement of the physical quantity. When high frequency noise (usually characterized by spikes or blurring) are smoothed by the filter it is called a low pass filter (LPF) because low frequencies are passed through, while high frequencies are rejected. If the signal from a sensor such as an electret microphone (shown in Example 3.8) contains a high frequency waveform, then a high pass filter (HPF) might be needed to reduce lower frequency signals, such as 60 Hz noise from a transformer or even the DC offset that is introduced from electret microphone DC biasing circuitry. If both high pass filtering and low pass filtering is needed on the same signal, then a band pass filter (BPF) is used. Filtering that occurs prior to the ADC can be done with hardware, but **filtering in software** after the signal is digitized is also a common practice. Using the filtering express VI in LabVIEW is an easy way to implement a software filter. The GUI for the Filter express VI is shown in Figure 3.6a. When **implementing a filter with hardware**, either passive or active filtering is used. A passive filter can be created using a resistor along with capacitors and/or inductors (RLC circuits) to reduce the amplitude level of noise (or any unwanted signal), while allowing the wanted signal to pass through. Passive filtering is described in **Module 4 of the Davis AC Circuits eBook**. Active filters created with Operational Amplifiers (Op Amps) work like passive filters, but instead of just passing through the wanted signal they are also capable of amplifying it. A Texas Instrument document title: Filter Design in Thirty Seconds is a good reference document that shows the basics of active filtering.

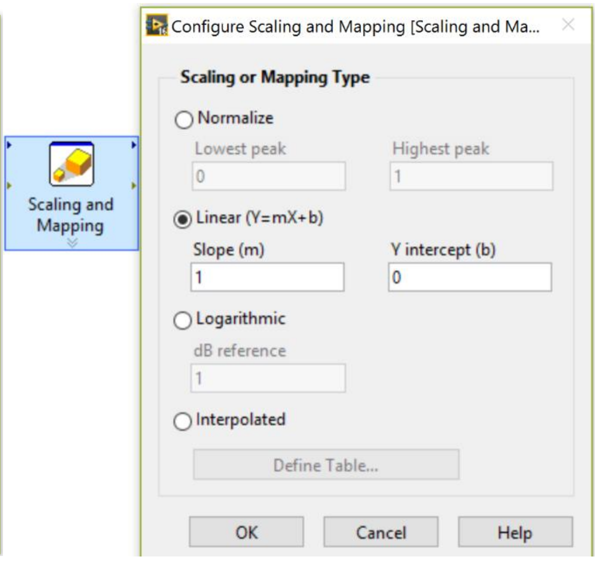
- Amplification or Attenuation – Sensors often have output levels that are very low and need to be amplified to a higher level so the signal can be used to adequately measure the physical quantity. When using a hardware amplification solution (instead of software), either Operational Amplifiers (**covered in Module 7 of the Davis AC Circuits eBook**) or transistors are usually used to amplify (i.e. increase the amplitude) a signal. Op Amps are especially useful in signal conditioning because they can easily provide both filtering and amplification when they are used as active filters. Less frequently, the sensor signal level might be too high to be compatible with the input of the DAQ device and attenuation is required. There are many options for implementing attenuation. Op Amps and transistors can also be used for attenuation, but a simple voltage divider circuit (adding 2 or more resistors in series) is the easiest to way to attenuate (i.e. reduce the amplitude) a signal. When performing amplification or attenuation in software, it is often referred to as scaling. The LabVIEW express VI called "Scaling and Mapping" can be used to accomplish this task. The GUI for this express VI is shown in Figure 3.6b. The terms amplification and attenuation usually imply that the signal is changed in a linear fashion such that Vout = G*Vin, where G is the amplification level (or voltage gain). If G < 1 then the signal is said to be attenuated. However, the LabVIEW scaling VI also allows non-linear types of scaling as shown in Figure 3.6b.



Filtering Type: Lowpass

Filter Specifications
Cutoff Frequency (Hz): 100
High cutoff frequency (Hz): 400
○ Finite impulse response (FIR) filter
Taps: 29
● Infinite impulse response (IIR) filter
Topology: Butterworth
Order: 3

View Mode
● Signals
○ Transfer function
☐ Show as spectrum
Scale Mode
☐ Magnitude in dB
☐ Frequency in log



Scaling or Mapping Type
○ Normalize
Lowest peak: 0
Highest peak: 1
● Linear (Y=mX+b)
Slope (m): 1
Y intercept (b): 0
○ Logarithmic
dB reference: 1
○ Interpolated

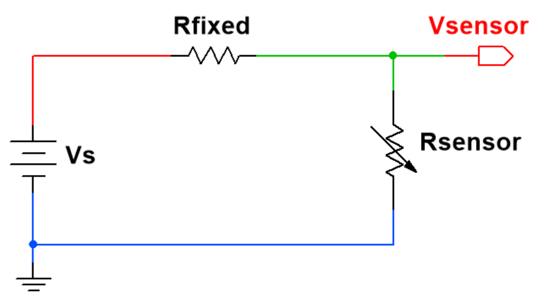
(a) (b)

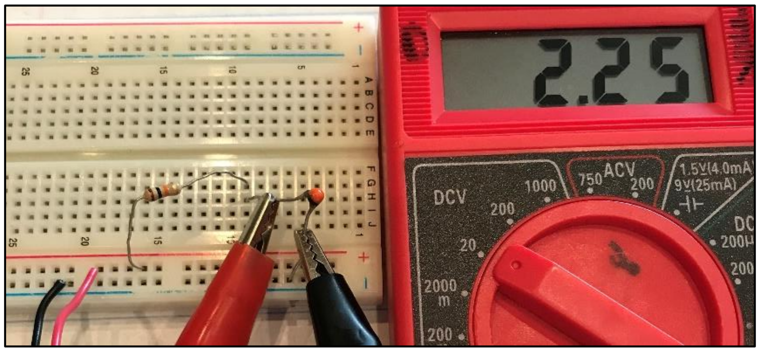
Figure 3.6: Express VI Examples, (a) Filter Express VI GUI, (b) Scaling and Mapping Express VI GUI

- Electrical Output Conversion – Another type of signal conditioning is required when the electrical output of the type of sensor is not compatible with a DAQ device or signal conditioning module. One example is a sensor with a current output (e.g. a photo diode) can have the current converted to a predetermined voltage range by adding a resistor network. There are also many sensors that have an electrical output of resistance (as shown in Module 4) and "excitation" and/or "bridge" circuitry are usually needed. Sensors that have a resistance output are often called "bridge-based" sensors. Excitation involves adding a power source so current flows through the sensor and the resistance output is converted to voltage using Ohm's Law. Bridge circuitry is described in more detail in Section 3.4.1 and examples using bridge circuitry for RTDs and strain gauges are shown in Section 4.1.2 and Section 4.4.1, respectively.

Section 3.4.1 – Converting Resistance to Voltage and Wheatstone Bridge Overview

This section is included because many sensors (such as the thermistor in the figure below) have an electrical output of resistance and this output usually needs to be converted to voltage. The easiest way to convert a sensor with an electrical output of resistance to a voltage is to put the sensor in the voltage divider configuration, as shown in Figure 3.7. From Equation 3.3, the location of R_{sensor} and R_{fixed} in the circuit below can be swapped to change the Vsensor output voltage from $\mathbf{Vs \cdot R_{sensor} \cdot (R_{sensor} + R_{fixed})}$ to $\mathbf{Vs \cdot R_{fixed} \cdot (R_{sensor} + R_{fixed})}$.





(a) (b)

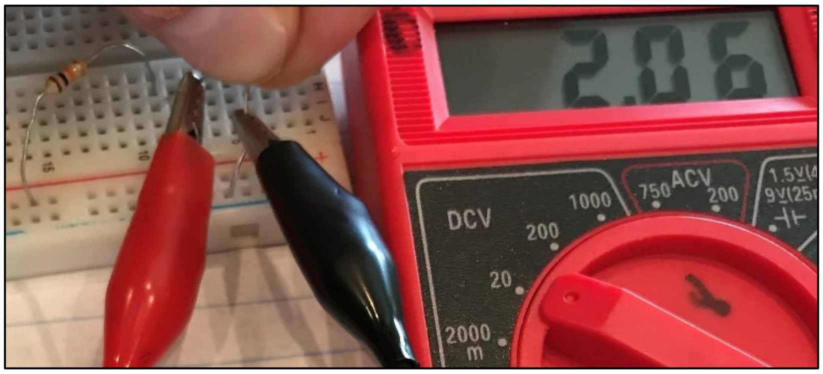
Figure 3.7: (a) Converting resistance to voltage using a voltage divider configuration. (b) Photo with a 10 kΩ fixed resistor and a 10 kΩ NTC thermistor in a voltage divider configuration.

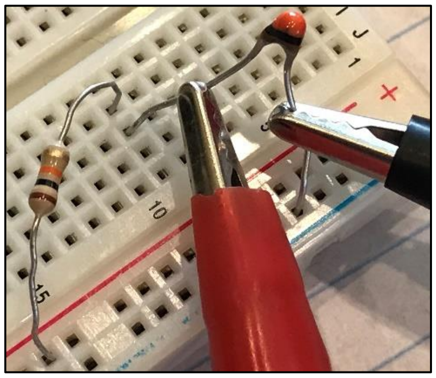
First, the voltage source in Figure 3.7 provides the current excitation needed to energize the sensor. Next, the output is taken between the resistor and the sensor (that acts like a temperature controlled resistor), so that the voltage changes as the sensor resistance changes according to the following equation (this is called the Voltage Divider Rule):

[3.3] $V_{Sensor} = V_S \cdot R_{Sensor} / (R_{Sensor} + R_{Fixed})$ **Voltage Divider Rule:** Equation [1.16] in the ***DC Circuits*** book

In the example in Figure 3.7, the voltage source was a 3-AA battery pack with a Vs of approximately 4.5V. Since the thermistor and fixed resistor were both the same values the voltage was divided by 2 from Equation 3.3.

The photo in Figure 3.7 is taken at room temperature (~ 72 ^oF). Figure 3.8a shows the thermistor being heated up to ~ 93 ^oF, which is the approximate value I got when taking the temperature at **my** fingertips (**this will vary**). When heated up, the thermistor's resistance drops and the voltage is reduced according to Equation 3.3.





(a) (b)

Figure 3.8: (a) Heating up thermistor in circuit in Figure 3.5. (b) Close up of voltage divider circuit.

An equation for Rsensor can be derived from Equation 3.3 as follows:

- From Equation 3.3 → $V_{Sensor} = V_S \cdot \mathbf{R}_{Sensor} / (\mathbf{R}_{Sensor} + R_{Fixed})$
- $V_{Sensor}/Vs = (\mathbf{R}_{Sensor}/(\mathbf{R}_{Sensor} + R_{Fixed}))$
- $(V_{Sensor}/Vs)^* (\mathbf{R}_{Sensor} + R_{Fixed}) = \mathbf{R}_{Sensor}$
- $(V_{Sensor}/Vs)^*\mathbf{R}_{Sensor} + (V_{Sensor}/Vs)^*R_{Fixed} = \mathbf{R}_{Sensor}$
- $(V_{Sensor}/Vs)^*R_{Fixed} = \mathbf{R}_{Sensor} - (V_{Sensor}/Vs)^*\mathbf{R}_{Sensor}$
- $(V_{Sensor}/Vs)^*R_{Fixed} = \mathbf{R}_{Sensor} {}^*[1 - (V_{Sensor}/Vs)]$
- $\mathbf{R}_{Sensor} = R_{Fixed}{}^*(V_{Sensor}/Vs) / [1 - (V_{Sensor}/Vs)]$

[3.4] $$R_{Sensor} = \frac{R_{Fixed} \cdot V_{Sensor}/V_S}{1 - V_{Sensor}/V_S}$$ Solved for R_{Sensor} from Equation 3.3

Example, Vs = 4.5V, R_{fixed} = 10 kΩ, and V_{sensor} = 2.06V, then $\mathbf{R}_{Sensor}$ = (2.06/4.5) · 10 kΩ/[1 – (2.06/4.5)] = **8.44 kΩ**

VDR Rule of Thumb: *Set the fixed resistor close to the middle resistance value of the sensor (i.e average of max and min expected resistance values) to get the most voltage change in a voltage divider configuration. For example, if a thermistor has a resistance of 10 kΩ at 72 ^oF and a resistance of 8.44 kΩ at 96 ^oF and that is the range of temperature you are expecting, then selecting a Rfixed resistor as close as (10 kΩ + 8.44 kΩ)/2 as possible would give you the most change in voltage versus change in resistance. This is related to the sensitivity concept discussed previously. If a sensor with a resistance output has a good sensitivity (i.e. large change in electrical output versus the change in physical property it is measuring) you can reduce that sensitivity as you convert the resistance to voltage by selecting a bad value of Rfixed so that the voltage change is limited.*

For example, if you select Vs = 4.5V and Rfixed to be 1 MΩ and the Rsensor value changes from 10 kΩ at room temperature to 8.44 kΩ at 96 ^{o}F, then the voltage values (calculated from equation 3.3) are equal to:

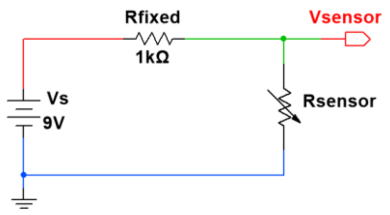
V_{sensor}(<u>72^{o}F</u>) = 4.5(10 kΩ/(10 kΩ + 1 MΩ) = <u>44.6 mV</u> & V_{sensor}(<u>96^{o}F</u>) = 4.5*(8.44 kΩ/(8.44 kΩ +1MΩ) = <u>37.7 mV</u>*

Conversely, if the optimal value of 9.22 kΩ (obtained from (10 kΩ + 8.44 kΩ)/2) was selected the results would be: V_{sensor}(<u>72^{o}F</u>) = 4.5(10 kΩ/(10 kΩ +9.22kΩ) = <u>2.34V</u> & V_{sensor}(<u>96^{o}F</u>) = 4.5*(8.44 kΩ/(8.44 kΩ +9.22kΩ) = <u>2.15V</u>*

This example shows that the change in voltage increases from ΔV = 44.6 – 37.7 = 6.9 mV when selecting a poor choice of a 1 MΩ resistor for Rfixed, to ΔV = 2.34 – 2.15 = 0.19 V = 190 mV when selecting the resistor for Rfixed based on the VDR Rule of Thumb. This results in a voltage range (ΔV) that is 26.5 larger, (190-6.9)/6.9 = 26.5, and therefore also results in a sensitivity (change in voltage versus change in temperature) that is also 26.5 larger.

If the voltage divider configuration is used with resistor-based sensors, precisely determining the temperature (or other physical quantity depending on the sensor type) is likely not the focus. The voltage divider configuration is primarily used to trigger other circuits when the physical quantity being measured reaches a threshold that experimentally has been determined to be the correct level. For example, the thermistor circuit in Figure 3.7 could be connected to a CMOS switching circuit and used to power a fan when the voltage drops below a certain threshold. Another example is discussed in Section 4.2.1, where photo resistors are used to automatically trigger lights to illuminate when it gets dark outside.

Example 3.11) *If the following circuit was used and the sensor resistance changed from 95 Ω to 105 Ω, calculate the change in voltage. How would you improve the voltage range of this circuit? Verify your answer is correct by performing calculations.*



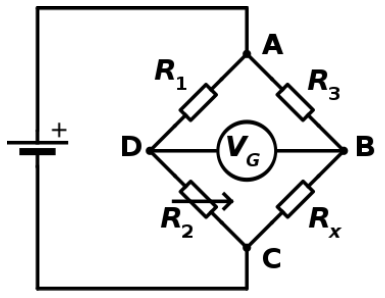
- When R_{Sensor} = 95 Ω → [3.3] V_{Sensor} = 9·(95/(**<u>1000</u>** + 95) = **<u>780.8 mV</u>**
- When Rsensor = 105 Ω → [3.3] Vsensor = 9·(105/(**<u>1000</u>** + 105) = **<u>855.2 mV</u>**
- The change in Vsensor = ΔV = 855.2 mV - 780.8 mV = **<u>74.4 mV</u>**

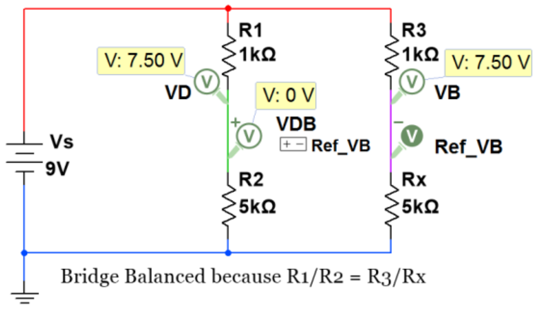
If Rfixed was sized according to the VDR Rule of Thumb then Rfixed should be set to **<u>100 Ω</u>** and the results are:

- When Rsensor = 95 Ω → [3.3] Vsensor = 9·(95/(**<u>100</u>** + 95) = **<u>4.385 V</u>**
- When Rsensor = 105 Ω → [3.3] Vsensor = 9·(105/(**<u>100</u>** + 105) = **<u>4.61 V</u>**
- The change in Vsensor = ΔV = 4.61 mV – 4.385 mV = **<u>225 mV</u>**

By changing the fixed resistor to the value specified in the VDR Rule of Thumb, the change in voltage tripled. The larger the change in voltage the easier it is to trigger circuitry and to correlate voltage to the physical quantity that the sensor measures (i.e. better sensitivity). This is especially true if there is significant noise present, because the noise floor might be high enough that the sensor signal can't be distinguished signal from noise because they are at similar amplitudes.

A more accurate method to convert resistance to voltage is done by using two voltage dividers in a configuration known as a **Wheatstone bridge**. This configuration is often times used with RTDs (Section 4.1.2) and strain gauges (Section 4.4.1). The bridge is **balanced** (i.e. V_{DB} = 0V and I_G = 0A) if **R1/R2 = R3/Rx**.





(a) (b)

Figure 3.9: (a) Wheatstone bridge. © Rhdv. Used under CC BY-SA license : https://en.wikipedia.org/wiki/Wheatstone_bridge#/media/File:Wheatstonebridge.svg
(b) Multisim schematic of a balanced Wheatstone bridge circuit showing V_{DB} = 0 V because R1/R2 = R3/Rx.

The V_G symbol in the center of the bridge denotes a Galvanometer, which is a device that detect small changes in current. A voltmeter could also be used to detect the change of voltage. Or, the voltages V_D and V_B could be measured by a DAQ device and V_{DB} (i.e. V_D-V_B) is the differential (or output) voltage of the bridge. R2 is drawn as a variable resistor in Figure 3.9a because a popular practice (see the Manual Method) is to adjust R2 to balance the bridge and, in turn, determine the unknown sensor value (If R1 = R3 and V_{DB} = 0 V, then Rx equals R2).

Wheatstone bridge Implementations

1) **Manual Method** – With R1 and R3 in Figure 3.9a set to identical values, R2 is specified as a precision variable resistor that is adjusted until the bridge is balanced at V_{DB} = 0 V. Since the bridge is balanced and R1 is equal to R3, then Rx is equal to the known value of the variable resistor. The variable resistor must have a good enough resistance resolution to obtain the desired Rx measurement. If cost is not an issue, the manual method is best accomplished by purchasing a piece of equipment that has the desired specifications. For example, the Yokogawa brand Wheatstone bridge device (part # 2768) can measure resistances from 100 mΩ to 110 MΩ with an accuracy that ranges from 0.01 % to 0.05%.
2) **Quarter Bridge – Only one active sensor (Rx) is in the bridge and R1, R2, and R3 are *fixed resistors:** ***Note:*** *Inactive sensors with known resistance can be used instead of fixed resistors.* The quarter bridge is often used with Resistance Temperature Detectors (RTD), as shown in Figure 4.5a. Lead resistance is often added into the quarter bridge (and the half bridge as well) to improve the accuracy, as shown in Figure_4_5b.
3) **Half Bridge – Two active sensors on one leg of the bridge and two *fixed resistors on the other leg:** This configuration and the full bridge configuration (listed next) are primarily used with strain gauges (Section 4.4.1). In the half bridge, R1 and R2 are usually specified to be identical fixed resistors and R2 and Rx are typically selected as strain gauges. One way half bridges are frequently configured is to place one of the strain gauges to measure compression strain and the other to measure tension strain.
4) **Full Bridge – All four resistors in the bridge are active sensors:** More information about this configuration (and also the quarter and half bridge configurations) is provided in the following link: https://www.transducertechniques.com/wheatstone-bridge.aspx

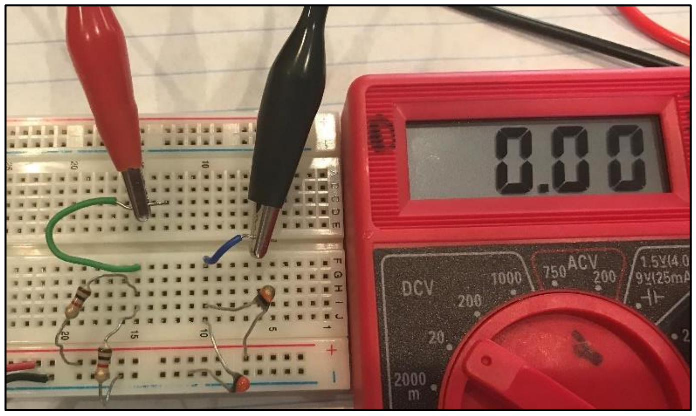
Anytime a Wheatstone bridge configuration is used and the values of R1, R2, and R3 are known, the resistance, Rx, can be solved for by using the V_{DB} measurement and equation 3.3. The following derivation shows the algebraic steps needed to convert equation 3.3 to equation 3.5 by solving for Rx.

- From equation 3.3: $V_D = Vs\cdot R2/(R1+R2)$ & $V_B = Vs\cdot \mathbf{Rx}/(R3+\mathbf{Rx})$
- $V_{DB} = V_D - V_B = Vs\cdot[R2/(R1+R2) - \mathbf{Rx}/(R3+\mathbf{Rx})]$
- $(V_{DB}/Vs) = [R2/(R1+R2) - \mathbf{Rx}/(R3+\mathbf{Rx})]$
- Find a common denominator: $(V_{DB}/Vs) = [R2\cdot(R3+\mathbf{Rx}) - \mathbf{Rx}\,(R1+R2)]/[(R1+R2)(R3+\mathbf{Rx})]$
- $(V_{DB}/Vs) = [R2\cdot R3+R2\cdot Rx- \mathbf{Rx}\cdot R1-Rx\cdot R2]/[R1\cdot R3+ R1\cdot \mathbf{Rx} +R2\cdot R3+R2\cdot \mathbf{Rx}]$
- $(V_{DB}/Vs)\,[R1\cdot R3+ R1\cdot \mathbf{Rx} +R2\cdot R3+R2\cdot \mathbf{Rx}] = R2\cdot R3 - \mathbf{Rx}\cdot R1$
- $(V_{DB}/Vs)\cdot[R1\cdot R3+R2\cdot R3]+ (V_{DB}/Vs)\cdot[R1\cdot\mathbf{Rx} +R2\cdot\mathbf{Rx}] = R2\cdot R3 - \mathbf{Rx}\cdot R1$
- $(V_{DB}/Vs)\cdot[R1\cdot R3+R2\cdot R3]+ (V_{DB}/Vs)\cdot(R1+R2)\cdot\mathbf{Rx} - R2\cdot R3 = - \mathbf{Rx}\cdot R1$
- $(V_{DB}/Vs)\cdot[R1\cdot R3+R2\cdot R3] - R2\cdot R3 = -\mathbf{Rx}\cdot R1 - (V_{DB}/Vs)\cdot(R1+R2)\cdot\mathbf{Rx}$
- $(V_{DB}/Vs)\cdot[R1\cdot R3+R2\cdot R3] - R2\cdot R3 = \mathbf{Rx}\cdot[-R1 - (V_{DB}/Vs)\cdot(R1+R2)]$

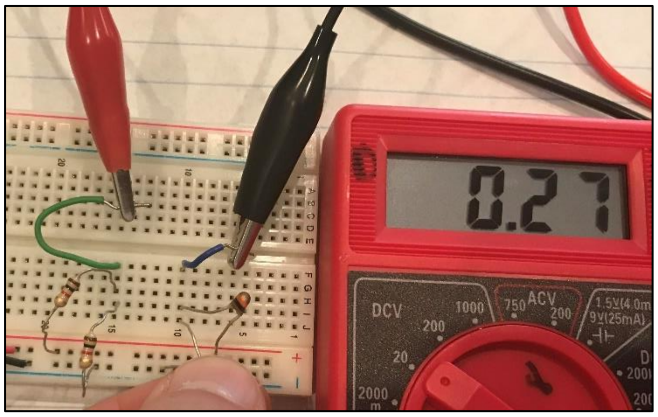
[3.5] $\mathbf{Rx} = [R2\cdot R3 - (V_{DB}/Vs)\cdot(R1\cdot R3+R2\cdot R3)]/[R1 + (V_{DB}/Vs)\cdot(R1+R2)]$ **Wheatstone Bridge circuit in Figure 3.9**

This is similar to the process used in the single voltage divider configuration, where equation 3.4 was derived to get the unknown sensor resistance. The main advantage of using bridge circuitry, instead of the voltage divider configuration, is that any noise in the circuit that is common to both V_D and V_B will cancel out. When trying to determine small changes in resistance, minimizing the noise can greatly increase the accuracy. When selecting the resistor values for the quarter bridge, the resistors in the left voltage divider leg of the Wheatstone bridge should be set equal (i.e. R1 =R2) and the same Rule of Thumb used for the voltage divider configuration should be used to set the size of R3. For quarter bridge strain gauge (Section 4.4.1) applications, R3 is often selected as an inactive (not loaded) gauge and Rx is the active (loaded) gauge. The quarter bridge configuration is demonstrated using thermistors in Example 3.12. R3 is "inactive" (i.e. It stays at ambient temperature and has a known resistance) and Rx is "active" (i.e. its temperature changes).

Example 3.12) *A Wheatstone bridge circuit is built with Vs = 4.5 V, R1 = R2 = 10 kΩ, and both R3 and Rx are thermistors that have a 10 kΩ resistance at a room temperature of 72 ^oF (See ohm meter measurement in Figure 3.11a). If Rx is heated up to ~ 93 ^oF by holding the device with your fingers (as shown in Figure 3.11b) and the bridge output voltage (VDB) is equal to 0.27 V (shown in Figure 3.10b), calculate the value of Rx using Equation 3.5. Measure the resistance with the ohm meter when holding the thermistor with your fingers (shown in Figure 3.11b) and calculate the % error between the calculated resistance and the measured resistance.*



(a)



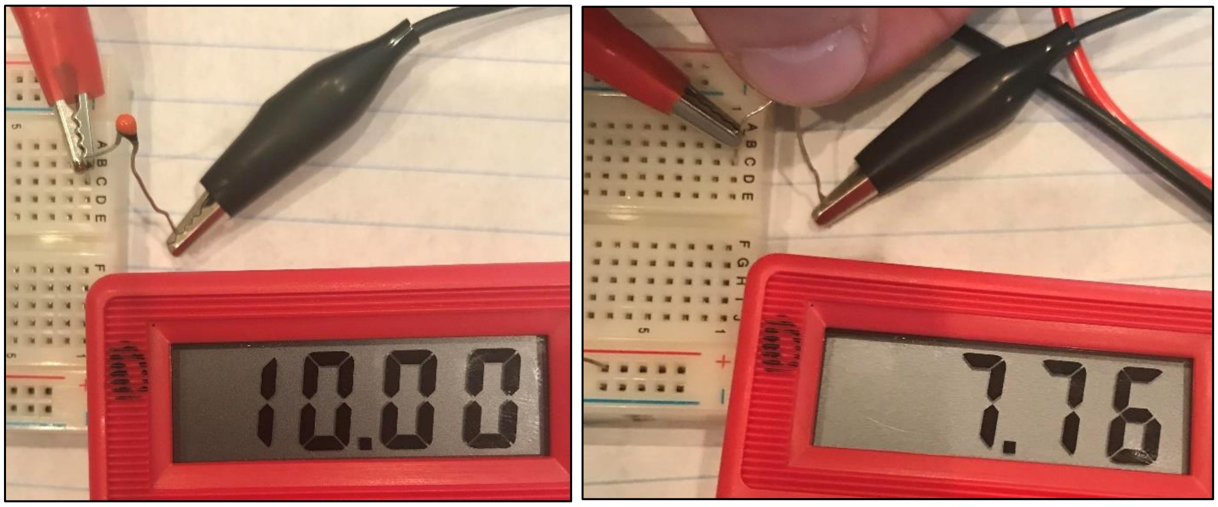
(b)

Figure 3.10: (a) Balanced bridge at ~ 72 ^oF (b) Unbalanced when thermistor, Rx, is heated to ~ 93 ^oF.

Plugging in values to solve for Rx:

[3.5] **Rx** = [R2·R3 - (V_{DB}/Vs)·(R1·R3+R2·R3)]/[R1 + (V_{DB}/Vs)·(R1+R2)]

- Rx = [10,000·10,000 - (0.27/4.5)·(10,000·10,000+10,000·10,000)]/[10,000 + (0.27/4.5)·(10,000+10,000)]
- **Rx = 7,857 Ω** (Figure 3.9a shows the measured resistance at ~ 93 ^{o}F is 7.76 kΩ).



(a) (b)

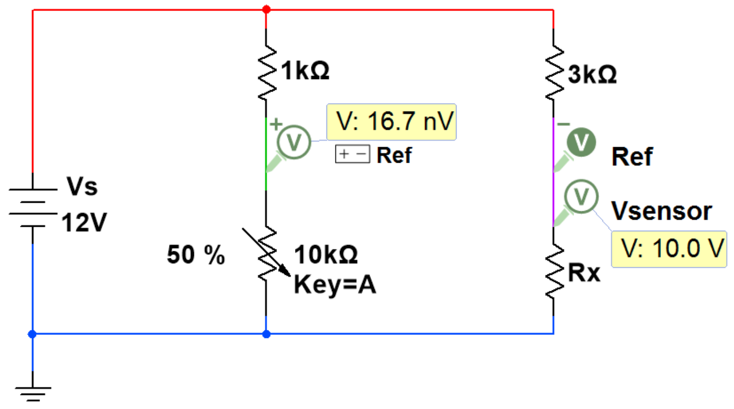
Figure 3.11: Meter units = kΩ, (a) 10 kΩ NTC Thermistor at ~ 72 ^{o}F (b) 10 kΩ NTC Thermistor at ~ 93 ^{o}F.

[3.1] **% Error = 100 · (Measured Value – True Value) / (True Value)**

% Error = 100 · (7.76 kΩ – 7.857 kΩ)/ 7.857 kΩ = **-1.24 %** (Difference between measured and calculated Rx)

There are numerous links in the Module 3 reference section (references 26 to 33) that provide additional details about the techniques and guidelines that are used when implementing Wheatstone bridges to increase the accuracy by removing noise and compensating for temperature effects. One example of an inaccuracy that occurs due to current excitation is **self-heating**. The best way to minimize self-heating effects is to keep the current that is flowing in the circuit as low as possible by limiting the size of the source voltage and being careful not to select bridge resistors that are too small. In addition to the following two examples, there are more examples involving Wheatstone bridges in the strain gauge section of the book.

Example 3.13) *The manual method described in the previous page is used to determine the unknown resistance, Rx, in the Wheatstone bridge circuit below. Use Rx to solve for the sensor voltage. Assume 16.7 nV is ~ 0V.*

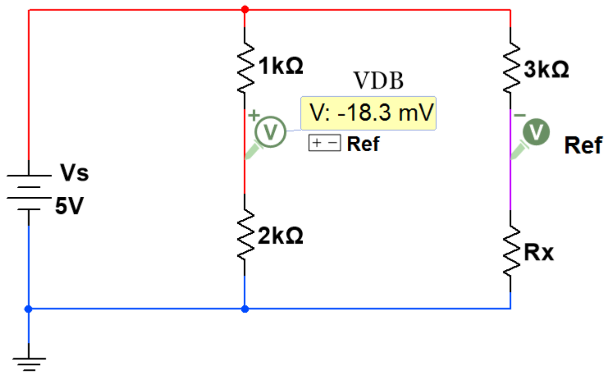


Solution

Balanced since VDB = ~ 0V
R1 = 1k, R2 = 0.5*10k = 5k
R1/R2 = 1/5 = 0.2
Balanced if R3/Rx = R1/R2 = 0.2
Rx = R3/0.2 = 3k/0.2 = 15k

The votlage across Rx can be solved using VDR
VRx = 12*(Rx/(Rx+3k)) = 12*15/(15+3) = 10 V

Example 3.14) *Determine the unknown resistance, Rx, in the Wheatstone bridge circuit below. Determine Rx and the sensor voltage.*



The bridge is Not Balanced since V_{DB} is not ~ 0V

R1 = 1 kΩ, R2 = 2 kΩ, R3 = 3 kΩ, Vs = 5V, VDB = -18.3 mV

Use Equation 3.5 to solve for Rx

* To make the calculations easier resistors will be put in kΩ.

$R_x = [R_2 \cdot R_3 - (V_{DB}/V_s)\cdot(R_1 \cdot R_3 + R_2 \cdot R_3)]/[R_1 + (V_{DB}/V_s)\cdot(R_1 + R_2)]$

$R_x = [2\cdot 3 - (-0.0183/5)\cdot(1\cdot 3+2\cdot 3)]/[1 +(-0.0183/5)\cdot(1+2)] =$ **<u>6.1 kΩ</u>**

Module 3 – References and Links

The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[1] Types of error info: https://en.wikipedia.org/wiki/Observational_error

[2] Basic Definitions: https://en.wikipedia.org/wiki/Accuracy_and_precision

[3] Basic Definitions: https://www.mccdaq.com/TechTips/TechTip-1.aspx

[4] Digital weight scale sensor (Spark Fun SEN 10245): https://www.sparkfun.com/products/10245

[5] Video showing how to make your own digital scale. http://www.nerdkits.com/videos/weighscale/

[6] Different types of resolution: https://en.wikipedia.org/wiki/Resolution

[7] Info about the 4 to 20 mA instrumentation standard: https://en.wikipedia.org/wiki/Current_loop

[8] Sampling Overview: http://download.ni.com/evaluation/pxi/Acquiring_Analog_Signal.pdf

[9] Sampling and Analog to Digital Conversion Overview: https://www.allaboutcircuits.com/technical-articles/understanding-analog-to-digital-converters-deciphering-resolution-and-sampl/

[10] Aliasing Overview and image used in Figure 3.2b: https://en.wikipedia.org/wiki/Aliasing

[11] Lego Mindstorm NXT and a comparison with the newer Lego EV3: http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/

[12] DAQ overview from NI: http://www.ni.com/data-acquisition/what-is/

[13] DAQ overview from Wikipedia: https://en.wikipedia.org/wiki/Data_acquisition

[14] USB specifications: https://en.wikipedia.org/wiki/USB

[15] Getting Started with Low-Cost DAQ: http://www.ni.com/tutorial/52241/en/

[16] Simultaneous Sampling Data Acquisition Architectures: http://www.ni.com/white-paper/4105/en/

[17] myDaq counter intro: http://digital.ni.com/public.nsf/allkb/347064BB2D9F168686257760004F06BF

[18] NI-DAQmx Express VI Tutorial: http://www.ni.com/tutorial/2744/en/

[19] Complete Guide to Building a Measurement System with

[20] the following sensors included (Temperature, Strain, Sound, Vibration, Position and Displacement, Pressure, Force): http://download.ni.com/evaluation/daq/Measurement_System_Build_Guide.pdf

[21] Frequency Measurements: How-To Guide: http://www.ni.com/tutorial/7111/en/

[22] Electret Microphone overview: https://en.wikipedia.org/wiki/Electret_microphone

[23] Parallax 180^0 Servo motor datasheet: https://www.parallax.com/product/900-00005

[24] Texas Instrument document titled "*Filter Design in Thirty Seconds*" that explains the basics of active filtering: http://www.ti.com/general/docs/litabsmultiplefilelist.tsp?literatureNumber=sloa093

[25] Bridge-Based Sensor Overview: http://zone.ni.com/reference/en-XX/help/370466AD-01/measfunds/bridgesensors/

[26] Signal Conditioning information link from the Bridge-Based Sensors Overview:

[27] http://zone.ni.com/reference/en-XX/help/370466AD-01/measfunds/signalconstrain/

[28] Bridge Excitation information link from the Bridge-Based Sensors Overview: http://zone.ni.com/reference/en-XX/help/370466AD-01/measfunds/bridgeexcitation/

[29] Hoffman, *Applying the Wheatstone Bridge Circuit*, http://eln.teilam.gr/sites/default/files/Wheatstone%20bridge.pdf

[30] Hoffman, *An Introduction to Measurements Using Strain Gauges*, http://www.kk-group.ru/help/Strain_Gauge_Measurements_Book_2012_01.pdf

[31] Bridge Measurement Systems: http://www.ti.com/lit/ml/slyp163/slyp163.pdf

[32] Omega Strain Gauge Overview: https://www.omega.com/literature/transactions/volume3/strain.html

[33] Strain Gauges and Whetstone bridges: https://www.transducertechniques.com/wheatstone-bridge.aspx

Module 4 – Sensors

Being able to automatically get measurements of physical properties into a computer is only possible with the use of sensors. Another reason why sensors are so important to our daily lives is that they are required to convert unintelligent open loop systems into closed loop (or feedback control) systems. An example of open loop systems are air conditioning units that are commonly found in cheap hotel rooms that have 4 settings (Off, Low, Medium, High). If you have ever had the pleasure of staying in one of these hotels, the following scenario might bring back some bad memories.

1. You go to bed and the temperature is perfect in the room so you leave the air conditioner off since the vent is approximately 6 inches from your head.
2. Then, you wake up in the middle of the night burning up and deleriously press the high button and after tossing and turning finally get back to sleep.
3. Next, you wake up an hour later and you are borderline hypothermic so you turn the air conditioner off.
4. Return to step 1.

This silly scenario explains why a long time ago we realized we need to have intelligent control systems that act autonomously to make our lives easier. A clever solution to this temperature problem was found by adding a sensor called a bimetallic strip thermostat (see Figure 4.3). In more recent times, advanced thermostats were created that use semiconductor-based temperature sensor ICs and have microcontrollers with built in PID feedback control algorithms that provide much better performance.

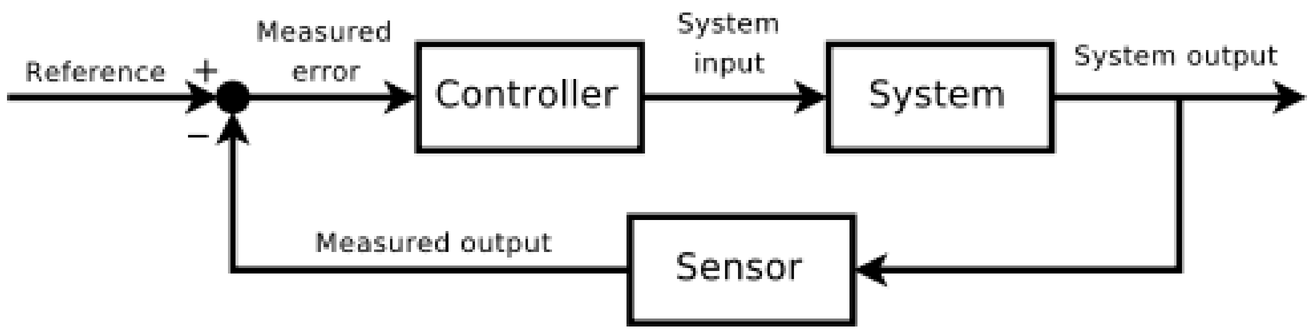


Figure 4.1) Feedback Control System. © Orzetto. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Control_theory#/media/File:Feedback_loop_with_descriptions.svg

The goals of this module are to provide an overview of many different types of sensors, give practical implementation information, and provide resources for further learning. While this module doesn't cover every type of sensor or every aspect of the sensors that are covered, it provides a good foundation for the most useful sensors that engineers deal with in industry. The types of sensors were carefully selected to provide a wide range of different types of operation so that learning about a new sensor type that is not in this book will likely be an extension of one of those covered herein and make it easier to figure out how to implement it.

In this book, sensors are broken into the following four categories and summary tables of the different types of sensors for each are listed at the beginning of each section. Practical examples of circuits built with many of these sensors are also also shown.

- Section 4.1 – Temperature Sensors
- Section 4.2 – Light Sensors
- Section 4.3 – Position Sensors
- Section 4.4 – Load or Strain Sensors

Section 4.1 – Temperature Sensors

There is, perhaps, more of a need to measure temperature than any other physical quantity. Without temperature measuring devices, your laptop would burn up, the heating and air conditioning in your house wouldn't work, you would have no idea if your car was about to catch fire from overheating, and many other critical aspects related to the proper operation and safety of equipment would break down. There are many types of sensors that measure temperature. The four main types of temperature sensors are shown in Table 4.1 and covered in the following sections: Thermocouples (Section 4.1.1), RTDs (Section 4.1.2), Thermistors (Section 4.1.3), and IC Temperature Sensors (Section 4.1.4). The first three types are called **passive** devices because they can't amplify a signal and need some sort of signal conditioning in order to be used in circuitry. The last sensor listed (IC temperature sensors) are usually **active** devices that only need to be connected to power and the signal conditioning is done internally in their integrated circuit design. References [1, 2, 3] provide an overview of these four types of temperatures sensors and some information in these sources is included in the following summary table.

Table 4.1) Temperature Sensors - Electrical output type and summary.

Sensor Type	input/output	Summary
Thermocouple	temperature/ voltage	Operation is based on the Seebeck Effect - Voltage across 2 different metals increases as temperature increases. Good for use in measuring devices, but not in circuitry. There are experimental results shown in Examples 7.12 and 7.13 for two different thermocouple types. Response times and sensitivities can be estimated from these plots. **Pros:** Inexpensive, rugged, and have the widest temperature range. **Cons:** Poor sensitivity and since they have a very low output voltage (see Figure 7.9), signal conditioning is needed. They are not as accurate as many other temperature sensors.
Resistance Temperature Detector (RTD)	temperature/ resistance	Resistance and temperature increase in metal wire, by approximately the following: $R2 = R1\cdot[1 + \alpha(T2-T1)]$. **Pros:** RTDs work in harsh environments, have best accuracy for wide temperature ranges. **Cons:** They are more expensive and sensitivity is not very good. They need current excitation, which leads to self-heating inaccuracies.
Thermistors	temperature/ resistance	Thermistors are semiconductor devices that change in resistance as the temperature changes. They usually have a negative temp coefficient (NTC type). There are numerous experiments using NTC thermistors in Section 3.4.1. **Pros:** They have the best sensitivity, are very inexpensive, can be very accurate if implemented properly, and can be integrated into circuitry fairly easily. **Cons:** They have a slow response time, very narrow temperature range, are highly nonlinear, and need current excitation to work which leads to self-heating inaccuracies.
IC Temperature Sensors	temperature/ current or voltage	They are semiconductor devices like thermistors, but have additional circuitry to prevent the need for signal conditioning. They are **active** devices so all that is needed is to be connected to a power supply. They can have either a current or voltage analog output or a digital output (depending on how the IC is designed). **Pros:** Have the best linearity, have very good accuracy (close to RTDs), and are the easiest to use in circuitry. **Cons:** More expensive than thermistors, and have a temperature range that is even narrower than thermistors.

In addition to the temperature sensors shown in Table 4.1 and discussed in more detail in sections 4.1 to 4.4, there are many other types of temperature sensing devices. Two of the more popular types of these are discussed below.

Infrared thermometers (also referred to as pyrometers) are popular type of temperature sensor that has the benefit of being a "non-contact" sensor. Since it doesn't make contact with the specimen, it has many applications such as measuring temperature of objects that are susceptible to contamination, moving, hard to access (or far away), or hazardous (high voltage, toxic environments, etc.). One common application is measuring the temperature coming out of vents to see if the air conditioning system is working properly (shown in Figure 4.2).

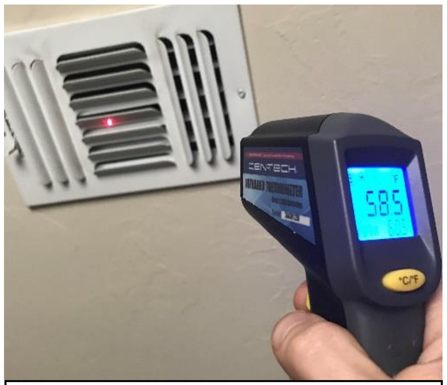


Figure 4.2: Measuring the temperature of the air coming out of a vent with an infrared thermometer (or pyrometer).

Details of the physics behind infrared thermometers are shown here. The previous link is from Omega.com. Omega is a great resource for purchasing or learning more about temperature sensors (as well as many other types of sensors) and will be used throughout this module as a resource. The following link from Omega is a very good practical introduction to infrared thermometers [6].

Bimetallic strips are another important type of temperature sensing devices that are used in traditional thermostats. They contain two different metals that expand at different rates as they are heated (as shown in Figure 4.3). By linking the ends together, the bimetallic strip can be positioned to move away from a contact and acts as a switch and control an air conditioning unit's operation.

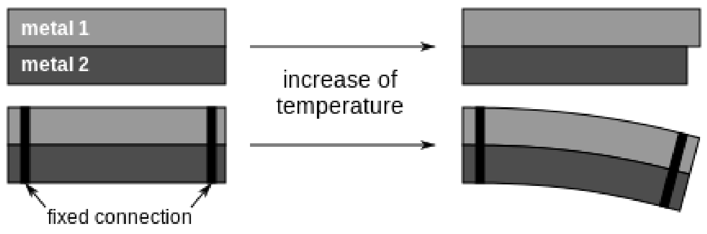


Figure 4.3: Bimetallic Strip temperature sensing device © Patrick87. Used under CC BY-SA License: https://en.wikipedia.org/wiki/Bimetal#/media/File:Bimetallic_stripe.svg

Section 4.1.1 – Thermocouples

In the bimetallic strip temperature sensing device, when two metals are bound together and heated up their different expansion rates lead to perpendicular movement. In addition to this mechanical property of two different metals that are bound together, there is also an electrical phenomenon that occurs across them called the Seebeck Effect. The Seebeck Effect causes a very small voltage to be generated across two different metals when there is a temperature difference between them. The Seebeck effect is the basis for how thermocouples work. Figure 4.5a shows the voltage versus temperature relationships of different thermocouple types. The relationship for some of them are very close to linear, but some aren't. If the relationship is assumed to be linear and the sensitivity of the thermocouple is known, equation 4.1 can be used to approximate the change in voltage versus change in temperature. As is the case with nearly all linear approximations, equation 4.1 is more accurate for small ranges (i.e. small difference between Tx and T_{REF}).

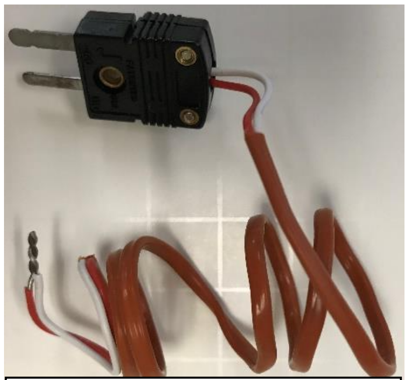


Figure 4.4: J-Type Thermocouple

[4.1] $Vx = k_T (Tx - T_{REF})$ Linear approximation thermocouple equation.

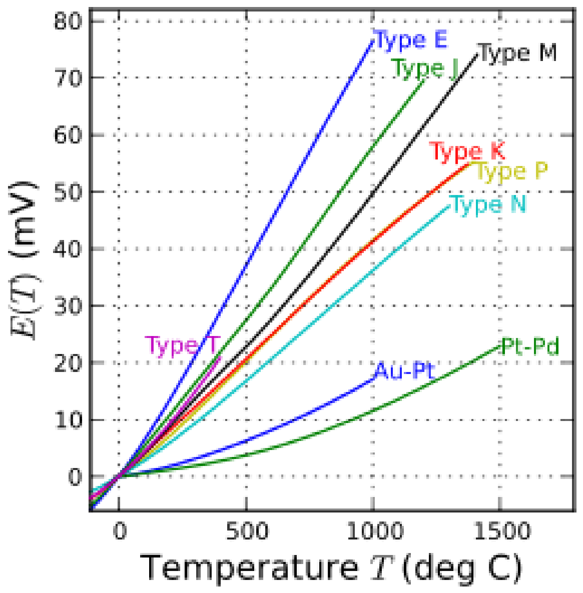
- k_T = Thermocouple **sensitivity** (shown in Table 4.2) = slope of the V vs T curve (shown in Figure 4.5a)

The units of kT set the units for the voltage (Vx) and temperature (Tx & T_{REF}). For example, if the units for the sensitivity, kT, is equal to µV/°C (like in Table 4.2) then the units for Vx are µV and the units for Tx & T_{REF} are °C.

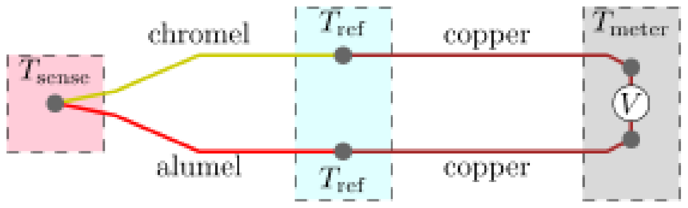
Each type of thermocouple has a different combination of metals that are used, which results in differences in operation and specifications. Table 4.2 shows some of the advantages and disadvantages of the four most common thermocouple types in terms of sensitivity, temperature range, and accuracy. Other things to consider when selecting a thermocouple type are cost, stability, longevity, oxidization, and magnetization.

Table 4.2) Common "Base Metal" Thermocouple Types. Sources: https://en.wikipedia.org/wiki/Thermocouple and https://www.omega.com/techref/colorcodes.html

Type	+ Wire Metal (Color)	- Wire Metal (Color)	Approximate Sensitivity, k_T	Temperature Range	Standard Limits of Error (Greater of these)
J	Fe (white)	Ci-Ni (red)	50 µV/°C	0 to 750°C	2.2°C or 0.75%
K	Ni-Cr (yellow)	Ni-Al (red)	41 µV/°C	-200 to 1250°C	2.2°C or 0.75%
E	Ni-Cr (purple)	Ci-Ni (red)	68 µV/°C	-200 to 900°C	1.7°C or 0.5%
T	Cu (blue)	Ci-Ni (red)	43 µV/°C	-250 to 350°C	1°C or 0.75%



(a)



(b)

Figure 4.5: a) Voltage versus Temperature Charts for Different Types of Thermocouples. Public Domain image: https://en.wikipedia.org/wiki/Thermocouple#/media/File:Intermediate_temperature_thermocouples_reference_functions.svg **b) K-Type thermocouple in the standard measurement configuration.** Public Domain Image: https://en.wikipedia.org/wiki/Thermocouple#/media/File:Thermocouple_circuit_Ktype_including_voltmeter_temperature.svg (Note: Ni-Cr = Chromel, Ni-Al = Alumel)

The standard way to take a temperature measurement with a thermocouple is shown in Figure 4.5b. The two ends of the thermocouple are connected together on the end that is in contact with the specimen being measured (Tsense) and the other end is placed at a known reference temperature (T_{REF} is ideally an ice bath that has a temperature of 0°C). Copper wire connects the reference ends of the thermocouple to the Multimeter or DAQ device so that a voltage measurement can be made. If the T_{REF} is not set at 0°C using an ice bath, then the temperature of the reference must be known by some other method, such as a secondary temperature sensor.

Example 4.1 shows how to calculate the temperature from a voltage measurement using equation 4.1 and compares the result to what would be obtained from a thermocouple reference table (a list of tables for each type of thermocouple is at the bottom of this link: https://www.omega.com/prodinfo/thermocouples.html).

Example 4.1) *A K-type thermocouple has a reading of 5.183 mV. If the reference temperature is at* 0^o C, what is the temperature, Tx, that corresponds to the thermocouple voltage reading?

[4.1] $Vx = k_T (Tx – T_{REF}) \rightarrow 5.183 \text{ mV} – 0 \text{ V} = 41\ \mu V/^oC (Tx – 0)$

Convert from mV to µV and solve for Tx → 5.183 mV = 5183 µV → Tx = (5183 µV)/(41 µV/^{o}C) = **126.4146 ^{o}C**

If the K-type thermocouple reference chart (https://www.omega.com/temperature/Z/pdf/z204-206.pdf) is used it shows the following values.

- V_{Low} = 5.165 mV @ 126 ^{o}C
- V_{High} = 5.206 mV @ 127 ^{o}C

Use linear interpolation to solve for Tx at a temperature between T_{Low} and T_{High}.

- $Tx = T_{Low} + (T_{High} – T_{Low}) \cdot (Vx - V_{Low}) /(V_{High} - V_{Low}) = 126 + (127 – 126) \cdot (5.183 – 5.165) /(5.206 – 5.165)$
- Tx = **126.439 ^{o}C**

Assuming the temperature reference chart contains the "true values", the calculation in this example using the linear equation 4.1 has the following error:

Equation 3.1 - % Error = 100 · (Measured – True) / (True) = 100 · (126.4146 - 126.439)/126.439 = **-0.0193 %**

The following example shows how to approximate the temperature using the ambient or room temperature as the reference and connecting the open end of the thermocouple directly to a Multimeter.

Example 4.2) *An experiment was recently performed where a K-type thermocouple was placed in the bend of my elbow and a reading of 0.42 mV was obtained on the Multimeter. A* ***thermometer*** *was used to measure the temperature at the bend of my elbow and it read* ***93.2 ^{o}F***. *The temperature in the room was measured to be 74 ^{o}F with a separate temperature sensing device and the open end of the thermocouple is connected directly to a Multimeter so that T_{REF} is assumed to be room temperature (~ 74 ^{o}F). Calculate the thermocouple temperature measurement and compare it to the measurement made with the thermometer.*

The first step is to convert mV to µV and ^{o}F to ^{o}C so that equation 4.1 can be used.

- Vx = 0.42 mV = 420 µV
- T_{REF} = 74 ^{o}F → T_{REF} _Celsius = (5/9)*(74 ^{o}F - 32 ^{o}F) = 23.33 ^{o}C

Equation [4.1] $\mathbf{Vx = k_T (Tx – T_{REF})}$ → 420 µV = 41 µV/^{o}C (Tx – 23.33 ^{o}C) → Tx = 33.574 ^{o}C

- Tx = 33.574 ^{o}C → Tx_Fahrenheit = 33.574^{o}C (9/5)+32 = 92.4332 ^{o}F **Error** = 92.4332 – 93.2 = **- 0.767 ^{o}C**
- Assuming the **thermometer** is the true value, the **% error** is 100·(92.4332 – **93.2**)/**93.2** = **- 0.823 %**

Example 4.2 shows how equation 4.1 works when the reference temperature is **NOT** at 0 ^{o}F. The temperature reference charts are specified at a reference temperature of 0 ^{o}C so a multiple step process must be followed (as shown in Example 4.3) when the reference temperature is **NOT** at 0 ^{o}C.

Note: *Figure 7.9 shows a plot of actual K-Type thermocouple readings with a voltmeter at room temperature. Figure 7.15 shows readings of a J-Type Thermocouple obtained from a Yokogawa temperature controller.*

Example 4.3) *Use the reference temperature chart to determine the unknown temperature in Example 4.2.*

If the K-type thermocouple reference chart (https://www.omega.com/temperature/Z/pdf/z204-206.pdf) is used it shows the following values for the reference temperature.

- V_{Low} = 0.919 mV @ 23 ^{o}C
- V_{High} = 0.96 mV @ 24 ^{o}C

Use linear interpolation to solve for Vx at a voltage between V_{Low} and V_{High}.

- $Vx = V_{Low} + (V_{High} – V_{Low}) \cdot (Tx - T_{Low}) /(T_{High} - T_{Low}) = 0.919 + (0.96 – 0.919) \cdot (23.33 – 23) /(24 – 23)$
- Vx = **0.93253 mV**

In order to use the temperature reference chart that is specified for 0 ^{o}C, the measurement of 0.42 mV needs be shifted up by 0.93253 mV.

- Vx = 0.42 mV + 0.93253 mV = 1.35253 mV

Now this value can be looked up in the temperature reference chart to determine Tx.

- V_{Low} = 1.326 mV @ 34 ^{o}C
- V_{High} = 1.366 mV @ 35 ^{o}C

Use linear interpolation to solve for Tx at a temperature between T_{Low} and T_{High}.

- $Tx = T_{Low} + (T_{High} – T_{Low}) \cdot (Vx - V_{Low}) /(V_{High} - V_{Low}) = 34 + (35 – 34) \cdot (1.35253 – 1.326) /(1.366 – 1.326)$
- Tx = **34.66325 ^{o}C**
- Tx = 34.66325 ^{o}C → Tx_^{o}F = 34.66325 ^{o}C (9/5)+32 = 94.39385 ^{o}F **Error** = 94.39385 – 93.2 = **1.194 ^{o}C**
- Assuming the thermometer is the true value, the **% error** is 100·(94.39385 – 93.2)/93.2= **+1.281 %**

The K-type thermocouple has an error of whichever is greater between 2.2^{o}C and 0.75%. In this case, 2.2 ^{o}C would be the greater value. The error calculations in both examples 4.2 and 4.3 have errors far less than 2.2^{o}C.

The following link provides a step by step guide to take a thermocouple measurement with a DAQ device in LabVIEW: http://www.ni.com/tutorial/14338/en/

The following Omega.com links provide additional information about thermocouples.

- Omega – Reference Junction Principles: https://www.omega.com/techref/thermoref.html
- Omega – Introduction to Thermocouples: https://www.omega.com/prodinfo/thermocouples.html
- Omega – Introduction to Thermocouple Wire: https://www.omega.com/prodinfo/thermocouplewire.html

Section 4.1.2 – Resistance Temperature Detectors (RTD)

The linear approximation of the relationship between temperature and resistance in metal wire is shown in Equation 4.2. More information about resistance change versus temperature was previously described in the resistance section of the Davis DC Circuits eBook.

[4.2] $R2 = R1 \cdot [1 + \alpha(T2-T1)]$

The temperature coefficient, α, is always a positive value for metals. A list of temperature coefficients for different types of metals and example problems showing methods to calculate the resistance of different materials at specified temperatures are shown in Section 1.1.2 of the Davis DC Circuits eBook. There are also equations that use multiple coefficients to better model the voltage verses temperature relationship for platinum RTDs. The Callendar-Van Dusen Equation, which is more accurate than equation 4.2, has 2 coefficients for temperatures above 0 ^{o}C and 3 coefficients for temperatures below 0 ^{o}C. RTDs are usually made with platinum, but less commonly Nickel or Copper RTDs are made that are cheaper and less accurate. The most common specification for platinum RTDs is the PT100 standard which means they are designed to have a 100 Ω resistance at 0 ^{o}C and a temperature coefficient of 0.00385 ($^{o}C^{-1}$) from 0 to 150°C. RTDs are also designed for different nominal resistances at 0 ^{o}C, such as the PT200 (200 Ω), PT500 (500 Ω), and PT1000 (1000 Ω) types. The following links show how the Callendar-Van Dusen equation and the DIN/IEC 60751 (also called IEC751) standard is used to obtain different accuracy classes for platinum RTDs [17, 18].

RTD Specification documents: https://www.omega.com/temperature/pdf/rtdspecs_ref.pdf and https://www.omega.com/toc_asp/frameset.html?book=Temperature&file=RTD_GEN_SPECS_REF

The four different platinum RTD classes listed in order from least to most accurate are shown below [17, 18]. The "T" is the variable used for true temperature and the units of T is ^{o}C.

• Class C = ±(0.60 + 0.01·T)	± 0.24Ω @ 0 ^{o}C	For temperatures between (-50 to 500°C)
• Class B = ± (0.30 + 0.005·T)	± 0.12Ω @ 0 ^{o}C	For temperatures between (-50 to 500°C)
• Class A = ± (0.15 + 0.002·T)	± 0.06Ω @ 0 ^{o}C	For temperatures between (-30 to 300°C)
• Class AA = ±(0.10 + 0.0017·T)	± 0.04Ω @ 0 ^{o}C	For temperatures between (0 to 150°C)

There are many different types of RTDs. For example, filtering by RTD in mouser under the temperature sensor product family yielded 397 products at the time of publication and the price ranged from as low as around $2 for a board mount PT100 sensor to over $100 for industrial RTD sensors designed for severe applications [19]. Thin Film RTD elements are another type of RTD product that is described in detail at the following link: https://www.omega.com/pptst/F1500_F2000_F4000.html

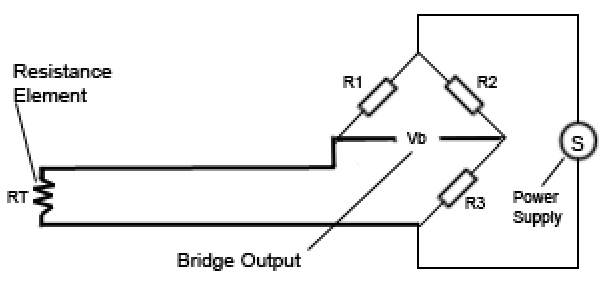
RTD Pros and Cons

RTDs (also called resistance thermometers) work in harsh environments and are much more accurate and stable than thermocouples, but on the contrary are more expensive, have a narrower temperature range, and have a much slower response time. Another negative characteristic of RTDs is that they need current excitation to operate (like all sensors that have a resistance output). Current excitation not only complicates the measurement system by adding the requirement of signal conditioning circuitry, but it can also lead to self-heating, which can result in temperature measurement inaccuracies. While thermistors and temperature sensor ICs (discussed in the next two sections) can have, in some situations, as good of accuracy as RTDs, their temperature range is extremely limited. RTDs aren't as easy to implement into circuitry as thermistors or IC temperature sensors, but are used much more frequently as temperature measurement probes.

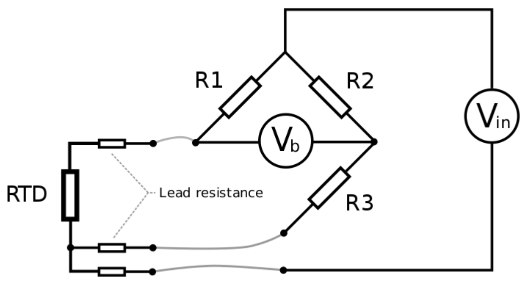
RTD Implementation

Figure 4.5a and Figure 4.5b shows the 2-wire and 3-wire RTD Wheatstone bridge configurations, respectively. The following link (http://www.ni.com/tutorial/14353/en/) shows how to make a RTD Measurement with a DAQ device in LabVIEW (**Note:** *The principles discussed in the NI tutorial can also be applied to taking thermistor measurements*). The previous NI link also shows how to connect each RTD pin to the DAQ for 2-wire, 3-wire, and 4-wire configurations. The 2-wire and 3-wire configurations, as well as a 4-wire configuration (that is used to get even greater noise immunity), are shown at this link (https://en.wikipedia.org/wiki/ thermometer). As previously described in Section 3.4.1, the Wheatstone bridge signal conditioning circuits in Figure 4.5a and

Figure 4.5b use a power supply for current excitation and fixed resistors to convert the unknown RTD resistance to a voltage. Typically, R1 and R2 are set at the same value and R3 is set close to the middle of the expected resistance output of the RTD (as previously described in the VDR Rule of Thumb) so that as the resistive element changes resistor values, the voltage will change by a larger value (i.e. increased sensitivity). By going from a 2-wire set up to 3-wire or 4-wire the noise is reduced considerably and accuracy is increased.



(a)



(b)

Figure 4.6: (a) Two-wire RTD configuration. © Psanderson. Used under a GNU Free Documentation License: https://commons.wikimedia.org/w/index.php?curid=6030373 **(b) Three-wire RTD configuration.** © Billy Huang. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Resistance_thermometer#/media/File:RTD_3Wire.svg

Section 4.1.3 – Thermistors

Like RTDs, thermistors also operate from a mathematical relationship between resistance and temperature, but its relationship is nonlinear. While metals, which are used for RTDs, have positive temperature coefficients (i.e. $\alpha > 0$), thermistors are made from semiconductor materials that usually have negative temperature coefficients (commonly referred to as NTC). Two examples are shown in Figure 4.7 and the one on the left was used in Section 3.4.1 to demonstrate the voltage divider configuration in Figure 3.7 and Figure 3.8 and the Wheatstone bridge circuit in Example 3.12. As shown previously, as the temperature increases, the resistance across the two pins of the NTC thermistor decreases. With additional signal conditioning circuitry, this change in resistance can be used in systems that require temperature detection and control, such as turning on a fan in your computer when it gets too hot internally. While there are some PTC thermistors, the vast majority of them are the NTC type and that is the only ones that will be discussed in this section.

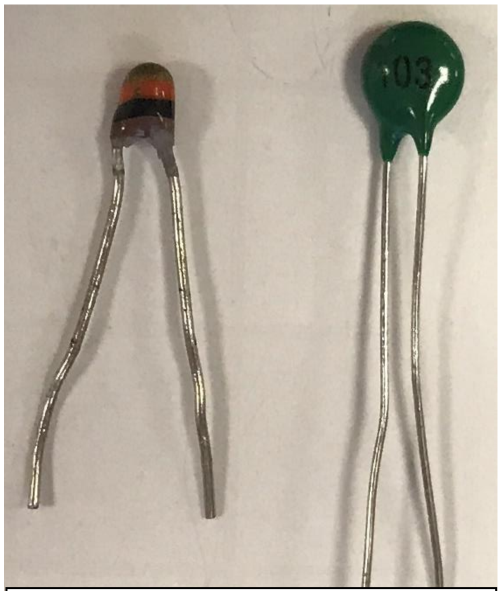
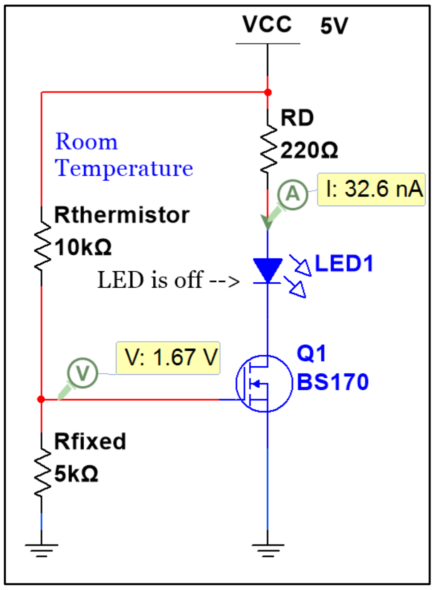
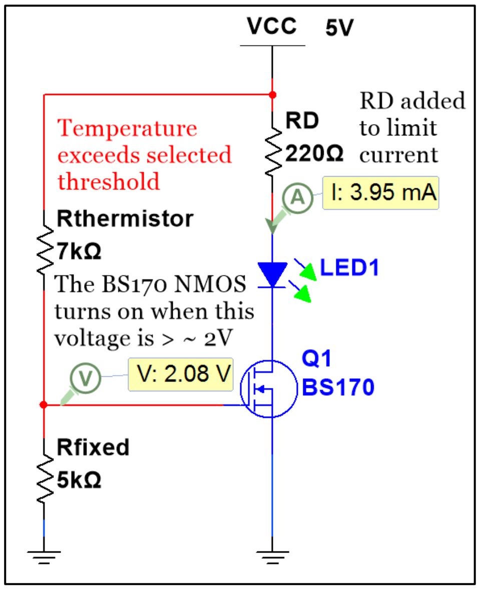


Figure 4.7: Two varieties of 10 kΩ NTC Thermistors (Link to the left one).

Thermistors can be accurate in some situations, but have many factors that can lead to inaccuracies. First, they have a much more nonlinear relationship between temperature and resistance as compared to thermocouples and RTDs, which makes them only usable for much small temperature ranges. A common temperature range for thermistors is around (-40 ^oC to 125 ^oC), which is much lower than thermocouples (-200 ^oC to 1450 ^oC) and RTDs (-260 ^oC to 850 ^oC) [1]. Their accuracy is also negatively affected when integrated into circuitry because they typically have a wide unit variation. Inaccuracy due to self-heating from the current excitation process is especially bad for thermistors because they are very small components and can be heated up easily.

The **biggest benefit** of thermistors is they are **small discrete components that can be easily used in circuitry**. Stabilizing the temperature for electronic circuits is one of their key applications. A voltage divider circuit similar to the one shown in Figure 3.7 and Figure 3.8 can be used to turn on a MOSFET switch. In Figure 4.8, a thermistor that is assumed to drop from ~ 10 kΩ at room temperature to ~ 7 kΩ at a temperature exceeds the limit for a hypothetical application. The following circuit shows how to turn on an LED when the temperature drops to 7 kΩ (as seen in Figure 4.8b). When the resistance is lower and the resistance is larger than 7 kΩ the LED is turned off (as seen in Figure 4.8a. One of the negative aspects of thermistors is their **slow response time**. As an example, when testing this thermistor it took approximately 15 seconds for the resistance to change from 10 kΩ to 7.76 kΩ in Example 3.12 when I covered it with my fingers to heat it up. Conversely, the **sensitivity is excellent** for thermistors. It changed from 10 kΩ to 7.76 kΩ as the temperature rose from 72 ^{o}F to 93 ^{o}F. This gives a sensitivity for this case of (10 kΩ - 7.76 kΩ)/(72 ^{o}F - 93 ^{o}F) = -106.7 Ω/^{o}F. Unfortunately, the thermistor is **highly non-linear** so this sensitivity value can't be used to accurately determine the resistance for a wide range of temperatures like was shown previously with sensors that are somewhat linear like thermocouples and RTDs.





VCC 5V
RD added to limit current
RD
220Ω
Temperature exceeds selected threshold
I: 3.95 mA
Rthermistor
7kΩ
LED1
The BS170 NMOS turns on when this voltage is > ~ 2V
V: 2.08 V
Q1
BS170
Rfixed
5kΩ

(a) (b)

Figure 4.8) Thermistor used to turn on LED when the temperature increases. (a) LED is Off, (b) LED is on.

The BS170 Metal Oxide Semiconductor Field Effect Transistor (MOSFET) is an N-channel device (NMOS) that has a threshold voltage of approximately 2 volts. This means that in the configuration shown above (and also in Figure 4.13 for the photo resistor) when the voltage in the yellow probe box exceeds 2 V, a significant amount of current flows through the LED. Thermistors are often used in a circuit like the one shown in Figure 4.8, but if the actual temperature needs to be determined from a resistance measurement then the temperature versus resistance relationship can be determined by different variations of the Steinhart-Hart equation.

The standard third order Steinhart-hart approximation is given in equation 4.3 [24].

$$[4.3] \quad \frac{1}{T} = A + B \cdot \ln(R) + C \cdot \ln{(R)}^3$$

There is also an extended version of the Stein-hart equation that includes more coefficients. The equation is shown on page 4 of the data sheet for the 3-color band style thermistor shown in Figure 4.6 [23].

The following link shows how to make a Thermistor Measurement with a DAQ device in LabVIEW. http://www.ni.com/white-paper/7112/en/

Section 4.1.4 – Temperature Sensor ICs

While thermistors (covered in Section 4.1.3) are discrete 2-pin semiconductor components that can be used to measure temperature in electronic applications, there are also other types of discrete semiconductor components that can be used. For example, the PN junction in semiconductor devices such as diodes and Bipolar Junction Transistors (BJT) have a small voltage changes when temperature changes (~ 2 mV/^{o}C). This voltage versus temperature relationship can be used in circuitry in the form of temperature sensor ICs. Temperature sensor ICs are semiconductor devices, like thermistors, but they have built-in signal conditioning circuitry so that they output voltage that changes linearly with temperature. There are too many different types of these sensors to cover in this book, but a few different types will be discussed. For the "through-hole" package style (ones that can be placed in a bread board instead of surface mounting to a PCB) they frequently come in the popular 3-pin Transistor Outline (TO) 92 package as shown in Figure 4.9a. In a 3-pin package, these active temperature sensors, will have power (Vcc) and ground as two of the pins and the output voltage as the other pin (see Figure 4.9b). The LMT85 is an example of an analog voltage output 3-pin TO-92 package and the DS1822+ is an example of a **digital output** 3-pin TO-92 package. Another popular discrete through-hole temperature sensor IC package has only 2-pins, as shown in Figure 4.9c. The ground pin from the 3-pin style package is left out and current flows out of the 2nd pin. The output is connected to an external "pulldown resistor" (it force the output to ground when no current flows) to convert current to voltage. An example of a 2-pin temperature senor IC is the AD590. Like the TO-92, surface mount versions of temperature sensor ICs usually only use 2 or 3 pins and all other pins in the IC package are not used (The unused pins are called "not connected" pins or NC on datasheets).

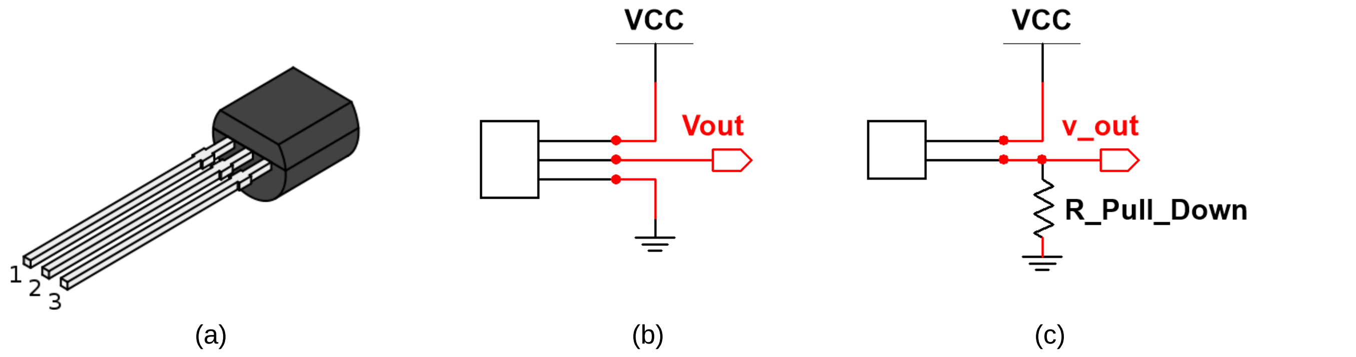


Figure 4.9: (a) TO-92 package. Public Domain: https://en.wikipedia.org/wiki/TO-92, **(b) 3-pin temperature IC, (c) 2-pin temperature IC, where an external "Pull Down" resistor must be added to convert current to voltage.**

- ❖ A **Pull Down resistor** forces a pin to go to ground when nothing is connected to that pin to avoid **floating**.
- ❖ A **Pull Up resistor** forces a pin to go to Vcc when nothing is connected to that pin to avoid **floating**.
- ❖ If a pin is **floating** it acts as an antenna and its value will fluctuate as it picks up noise.
- ❖ A **digital sensor** has internal circuitry that converts the output to 0V or Vcc, which usually allows them to be connected directly to a DAQ without using an ADC.

Like thermistors, temperature sensor ICs are small discrete components that can be easily integrated into circuitry. Thermistors required current excitation to convert the resistance output to voltage for them to be used in circuitry, but temperature sensor ICs are active devices that only need to be connected to power to get them to output current or voltage. Since no signal conditioning is needed, they are easier than thermistors to use in a circuit. They also are more accurate and are much more linear than thermistors. Their only real drawback as compared to thermistors is that they are more expensive and often have a narrower temperature range.

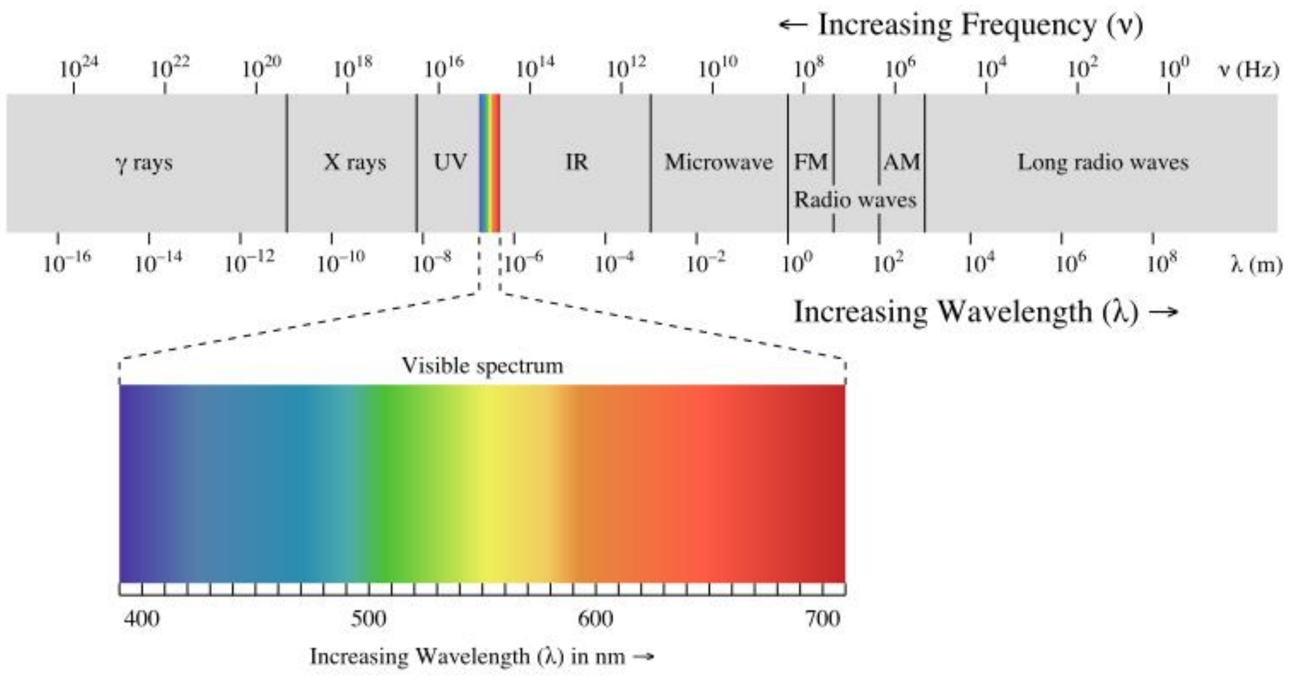
Section 4.2 – Light Sensors

Light sensors are generally divided into two types: photoconductive and photovoltaic. The photovoltaic sensors are further subdivided into photodiodes, phototransistors, and solar cells, as shown in Table 4.3.

Table 4.3) Light Sensors - Electrical output type and summary.

Sensor Type	input/output	Summary
Photoconductive sensor (CdS Cell)	light/ resistance	Also called photo resistors, light-dependent resistors (LDR), or CdS cells. CdS refers to Cadmium Sulfide (the material they are made from). When light increases across a CdS cell, the resistance decreases. They are cheap 2-terminal devices and easy to use in a voltage divider configuration. They have a limited spectrum and are usually used with visible light.
Photovoltaic sensors (photodiode)	light/current	Using the photovoltaic effect, photodiodes are semiconductor pn junctions that convert light into electrical energy. Photodiodes are small 2-terminal devices that have an increase in current when light increases. They are usually used with infrared light that has a higher wavelength than visible light.
Photovoltaic sensors (phototransistor)	light/voltage	Photodiodes and transistors are combined into one package to create a phototransistor, so that the current is amplified and converted to voltage. Phototransistors have a slower response time than Photodiodes.
Photovoltaic sensors (solar cells)	light/current	Solar cells are large area photodiodes that are optimized for to convert light to power efficiently. Multiple solar cells are combined to create solar panels. Systems created with solar panels are also referred to as Photovoltaic (PV) systems.

The primary factor that separates different types of light sensors is often related to the frequency spectrum, shown below. As wavelength (λ) goes up the frequency goes down, due to the equation **Frequency = c/λ**, where c is the speed of light and equal to approximately 299,792,458 m/s.

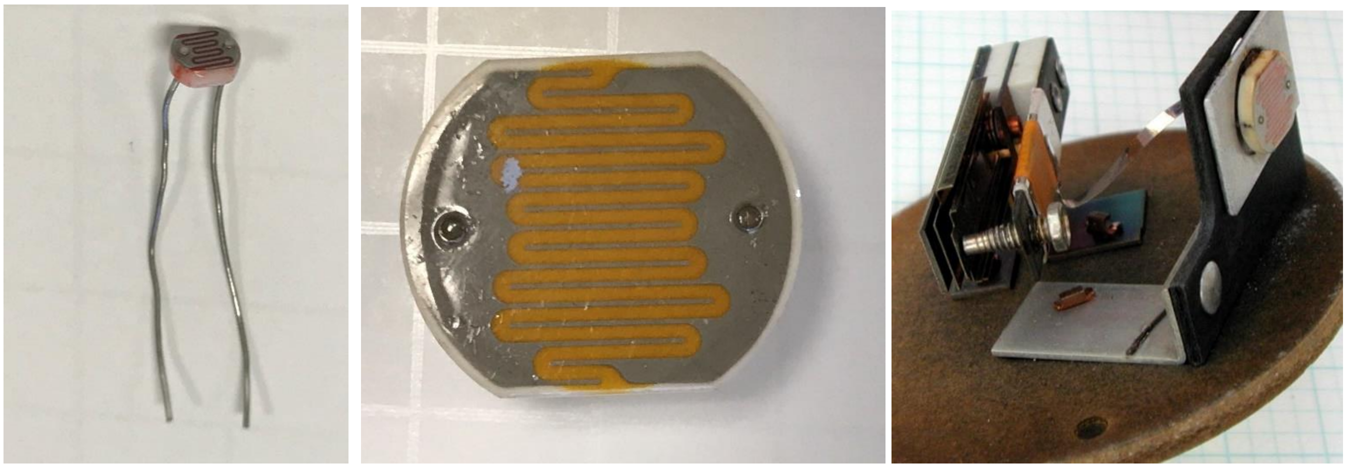


← Increasing Frequency (ν)
10^{24} 10^{22} 10^{20} 10^{18} 10^{16} 10^{14} 10^{12} 10^{10} 10^{8} 10^{6} 10^{4} 10^{2} 10^{0} ν (Hz)
γ rays
X rays
UV
IR
Microwave
FM
AM
Radio waves
Long radio waves
10^{-16} 10^{-14} 10^{-12} 10^{-10} 10^{-8} 10^{-6} 10^{-4} 10^{-2} 10^{0} 10^{2} 10^{4} 10^{6} 10^{8} λ (m)
Increasing Wavelength (λ) →
Visible spectrum
400 500 600 700
Increasing Wavelength (λ) in nm →

Figure 4.10: Electromagnetic spectrum. © Philip Ronan. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Light#/media/File:EM_spectrum.svg

Section 4.2.1 – Photoconductive sensors

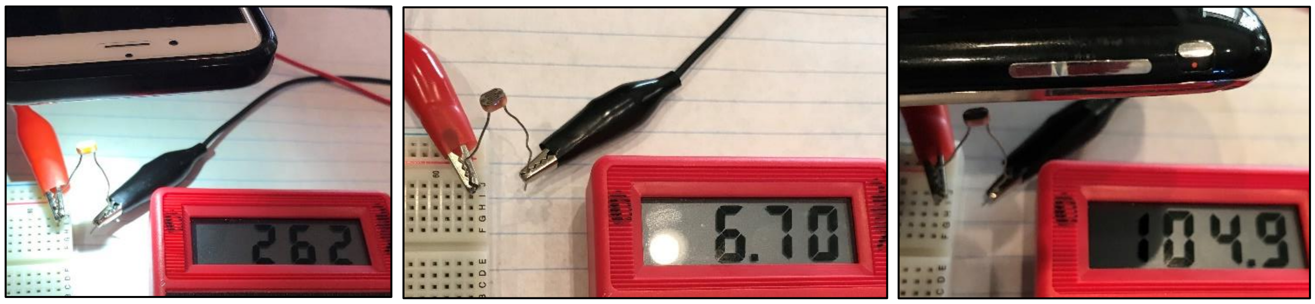
Photoconductive sensor are also called photo resistors, light-dependent resistors (LDR), or Cadmium Sulfide (CdS) cells. The resistance of these two terminal devices decreases as light increases. They have spectral range that is mostly limited to visible light (400 to 700 nm as shown in Figure 4.10) so they are not used with infrared light, like photodiodes (discussed next). Figure 4.11 shows how the resistance of a CdS cell changes with different levels of light. The photos in Figure 4.10 show three slightly different types of CdS cells. Photo (b) shows a close up of the face and photo (c) shows the CdS cell that is used to turn on a street light when it gets dark. For the remainder of this section, the term **CdS cell** will be used instead of photo resistor.



(a) (b) (c)

Figure 4.11: (a) Small CdS cells. (b) Larger CdS Cell – close up of face. (c) CdS Cell used to turn on a street light when it gets dark. © Atlant. Used under CC BY-SA: https://en.wikipedia.org/wiki/Photoresistor#/media/File:Streetlight_control.jpg

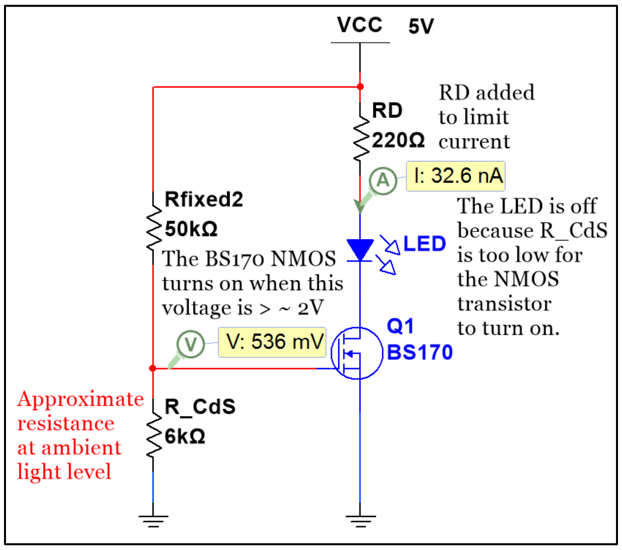
The resistance versus light relationship varies between CdS cells, but in general the resistance drops to a resistance in the hundreds of Ohms when light shines on the CdS cell and typically increases to well beyond 100 kΩ when it is dark. Figure 4.11 shows an experiment that uses the CdS cell in Figure 4.11a. The resistance changes from 262 Ω (Figure 4.12a) to 104.9 kΩ (Figure 4.12c) as the sensor is adjusted from being exposed to bright light to being covered so that it appears to be dark. If brighter light is shined on the CdS cell, it can drop below 100 Ω and when it gets darker it can increase in resistance to a much higher value. Since the resistance changes dramatically in a CdS cell (as demonstrated in Figure 4.12) it is considered to have good sensitivity.



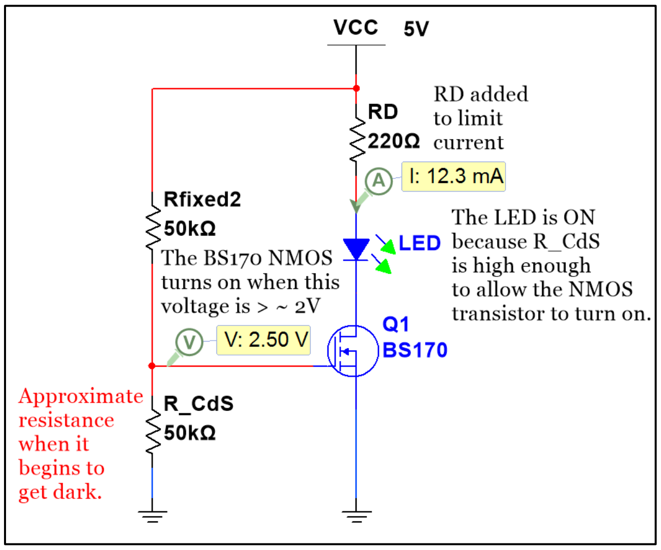
(a) (b) (c)

Figure 4.12: (a) R_{Cds} = 262 Ω with bright light (b) R_{Cds} = 6.7 kΩ with ambient light (c) R_{Cds} = 104.9 kΩ when dark

To covert the resistance of a CdS cell to a voltage, the voltage divider configuration shown originally in Figure 3.5 is typically used. Since the exact amount of light is usually not trying to be accurately measured, using a Wheatstone bridge circuit wouldn't make much sense with a CdS cell. Rather, different light levels are characterized by the approximate resistance seen by the CdS cell and used to control circuitry. Figure 4.11c shows a CdS cell inside a street light that is used to turn on the light automatically at night time. Figure 4.13 shows the circuit that could be used to detect night time and turn on a street light (or LED in this case). In this way, CdS cells are also frequently used to control other items, such as outdoor light bulbs or Christmas lights.



VCC 5V
RD 220Ω
RD added to limit current
I: 32.6 nA
Rfixed2 50kΩ
LED
The LED is off because R_CdS is too low for the NMOS transistor to turn on.
The BS170 NMOS turns on when this voltage is > ~ 2V
V: 536 mV
Q1 BS170
Approximate resistance at ambient light level
R_CdS 6kΩ

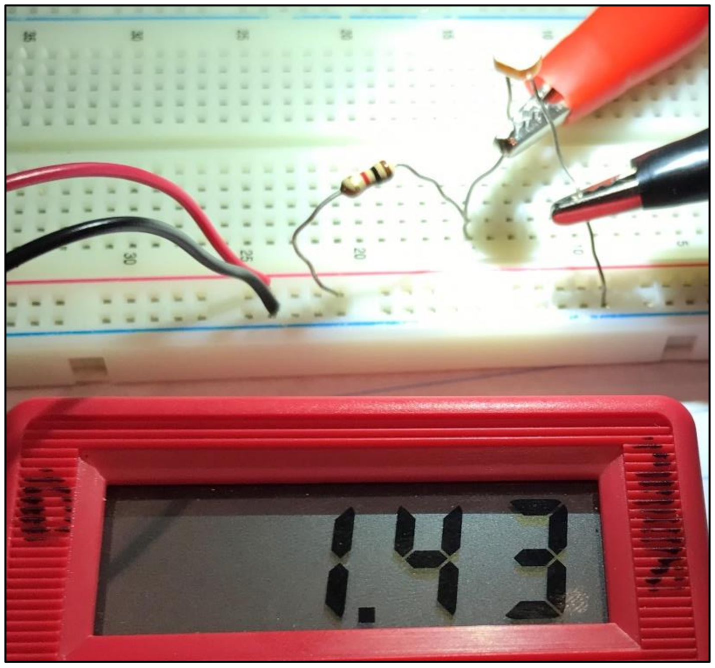


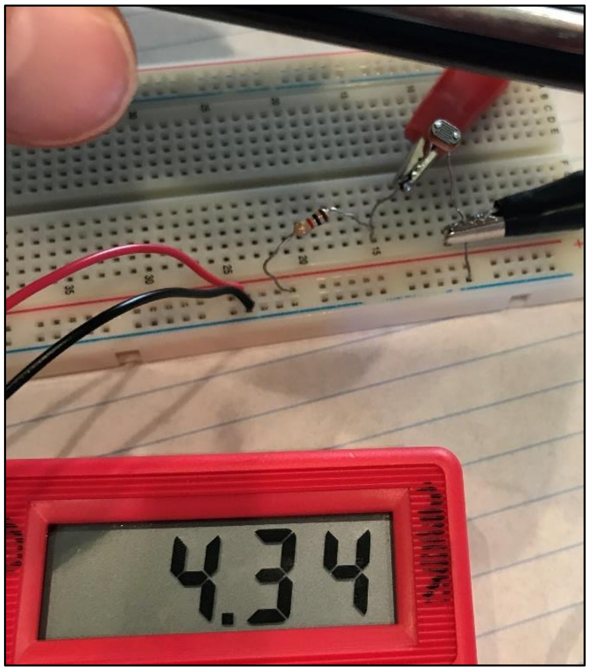
VCC 5V
RD 220Ω
RD added to limit current
I: 12.3 mA
Rfixed2 50kΩ
LED
The LED is ON because R_CdS is high enough to allow the NMOS transistor to turn on.
The BS170 NMOS turns on when this voltage is > ~ 2V
V: 2.50 V
Q1 BS170
Approximate resistance when it begins to get dark.
R_CdS 50kΩ

(a) (b)

Figure 4.13: (a) CdS cell circuit – LED off when sensing light. (b) CdS cell circuit – LED on when sensing dark.

A voltage divider circuit is built in Figure 4.14 to show how the CdS cell voltage increases when it gets dark.





(a) (b)

Figure 4.14: CdS cell VD Circuit (a): V_{CdS} = 1.43 V when sensing bright light. (b) V_{CdS} = 4.34 V when sensing dark.

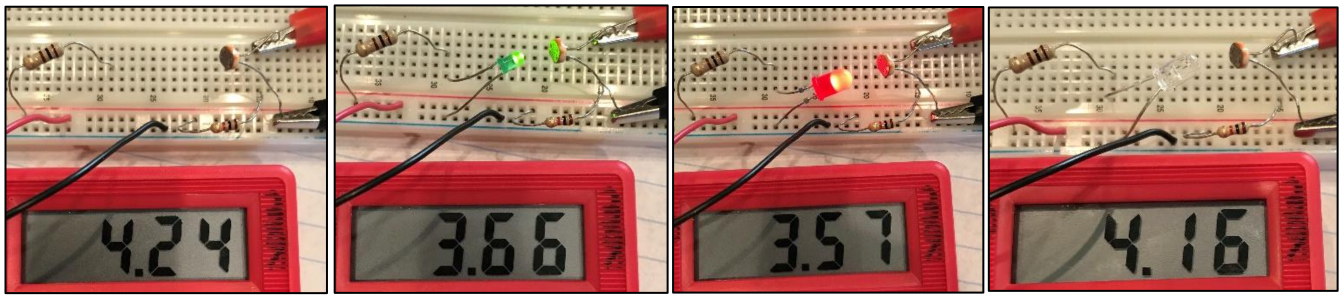
In the same way that the thermistor resistance was determined from Equation 3.4 for the circuit in Figure 3.6, the resistance of the CdS cell for light and dark can be determined from the voltages in Figure 4.14.

In Figure 4.14, the voltage source is 4.5 V, R_{Fixed} = 1 kΩ, and $V_{sensor} = V_{CdS}$. R_{Sensor} can be calculated as follows:

Recall Equation 3.4 is the following: $$R_{Sensor} = \frac{R_{Fixed} \cdot V_{Sensor}/V_S}{1 - V_{Sensor}/V_S}$$

- From Figure 4.14a V_{sensor} = 1.43 V → R_{Sensor} =[R_{Fixed}·(V_{Sensor}/Vs)]/[1–(V_{Sensor}/Vs)]=[1k·(1.43/4.5)]/[1 – (1.43/4.5)
  - ➢ R_{Sensor} = **466 Ω** (For part (a) when it senses bright light)
- From Figure 4.14b V_{sensor} = 4.34 V → R_{Sensor} =[R_{Fixed}·(V_{Sensor}/Vs)]/[1–(V_{Sensor}/Vs)]=[1k·(4.34/4.5)]/[1 – (4.34/4.5)
  - ➢ R_{Sensor} = **27,125 Ω** (For part (b) when it senses dark))

Like all Photo resistors, CdS cells have a limited spectral range so their applications are limited to visible light (wavelengths between 400 to 700 nm as shown in Figure 4.10). In Figure 4.15, different types of light are used on the CdS cell to determine which types of light the CdS cell most effectively responds (or is sensitive) to.



(a) Ambient Light (No LED) (b) Green LED (c) Red LED (d) Infrared (IR) LED

Figure 4.15: CdS cell experiment to determine response to different types of light.

The voltages (V_{Sensor}) in Figure 4.15 are converted to the sensor resistance (R_{Sensor}) using Equation 3.4 in the bullets below. First, it should be stated that the voltage source is 4.5 V and a) through d) all use a 1 kΩ fixed resistor in a voltage divider circuit. The ambient lighting is the control of the experiment and the CdS cell voltage in (a) depends on the light level in the room. By using the ambient lighting (no added LED) as the control, a comparison can be made of the change in resistance (ΔR) between ambient light conditions and the different LEDs. This will provide an example of how the CdS cell responds to the different wavelengths of light.

- (a) Control - Ambient light: V_{Sensor} = 4.24 V → R_{Sensor} = 16.31 kΩ
- (b) Green LED (Wavelength = 520 to 550 nm [32]): V_{Sensor} = 3.66 V → R_{Sensor} = 4.36 kΩ, **ΔR = -11.95 kΩ**
- (c) Red LED (Wavelength = 620 to 645 nm [32]): V_{Sensor} = 3.57 V → R_{Sensor} = 3.84 kΩ, **ΔR = -12.47 kΩ**
- (d) Infrared LED (Wavelength = 940 nm [34]) : V_{Sensor} = 4.16 V → R_{Sensor} = 12.24 kΩ, **ΔR = -4.07 kΩ**

The vastly different voltage and resistance changes when comparing the green and red visible light LEDs to the infrared LEDs explain why CdS cells are typically used only in the visible light region. Additionally, the voltage only changed from ambient lighting in photo (a) to IR LED lighting in photo (d) by 80 mV, which is not very far beyond the natural fluctuations (or noise) that was observed for the circuit in part (a). It also should be pointed out that the IR LED in photo (d) appears to be off. This is due to the fact that the iPhone 7 camera, that was used to take the photo, filters out infrared light. Figure 4.20 discusses this topic in more detail. The cover page of this book shows an example of both IR LEDs and bright white LEDs that are visible with a camera. The IR LED appears purple in the photo, but when observed with the human eye it appears to be off (as it looks in Figure 4.15d).

Section 4.2.2 – Photovoltaic sensors: Photodiodes

In general, photovoltaic sensors operate by the photovoltaic effect, where light falling on a semiconductor pn junction results in the generation of electrical energy. Photodiodes are the simplest type of photovoltaic sensors. They are two terminal devices that exhibit an increase in current as the light that is in contact with them increases. The figure below shows a photo of a photodiode and the circuit symbol used for them. Notice in the close up photo of the face of the photodiode, there is a transparent covering that protects the pn junction, but doesn't inhibit light so that the already low current output of the photodiode is not further reduced.

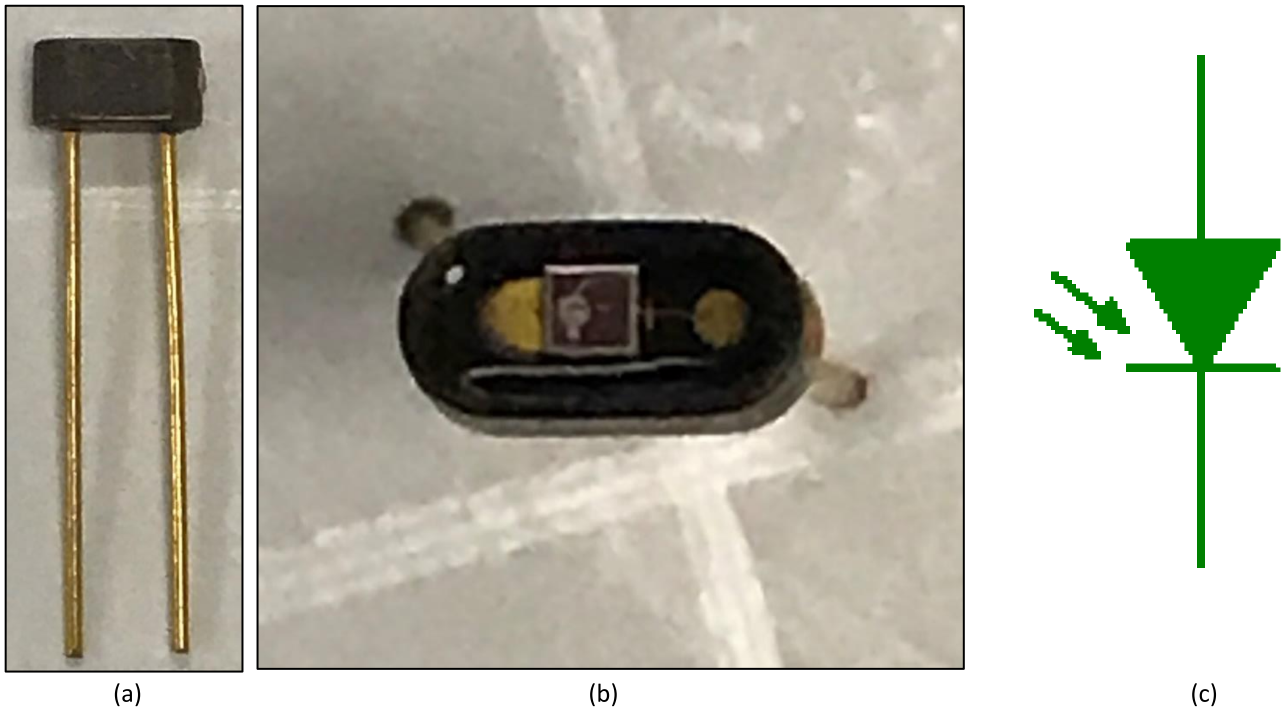


Figure 4.16: (a) Example of a Photodiode, (b) Close up of the face of photodiode, (c) photodiode circuit symbol

Unlike photo conductive sensors, discussed in Section 4.2.1, that work primarily with visible light, photodiodes are usually used in infrared light applications. The most common semiconductor materials that are used to make photodiodes are **Silicon** (190 nm < λ < 1100 nm), **Germanium** (400 nm < λ < 1700 nm), and **Indium Gallium Arsenide (GaAs)** (800 nm < λ < 2600 nm) [36]. The range of wavelength is given for each material, but the exact spectral sensitivity (also called response, which is equal to photo current divided by light power) will vary greatly depending on the design of the photodiode. Figure 4.17 shows an example of a Silicon photodiode that has a peak response at infrared wavelengths at around 940 nm. In contrast, Figure 2 of the Sharp PD49PI/481PI photodiode datasheet shows spectral sensitivity curves that are much narrower than the photodiode example in Figure 4.17. These Sharp photodiodes, like many others, are made to work with IR light and have almost no response for visible light (400 nm < λ < 700 nm).

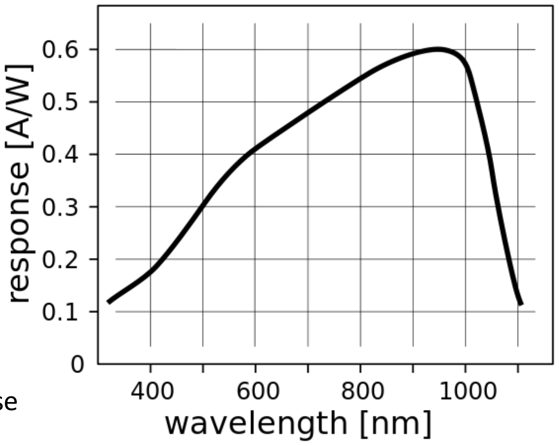


Figure 4.17: Response of a silicon photodiode vs wavelength of the incident light. © KaiMartin. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Photodiode#/media/File:Response_silicon_photodiode

A photodiode can be combined with a trans-impedance amplifier to convert the "photo current" output (i.e. current produced by the photodiode) to voltage, as shown in the figure below. Texas Instrument provides a good overview of transimpedance amplifier design [37] and Analog Devices has a very nice online wizard design tool for photodiodes [38, 39].

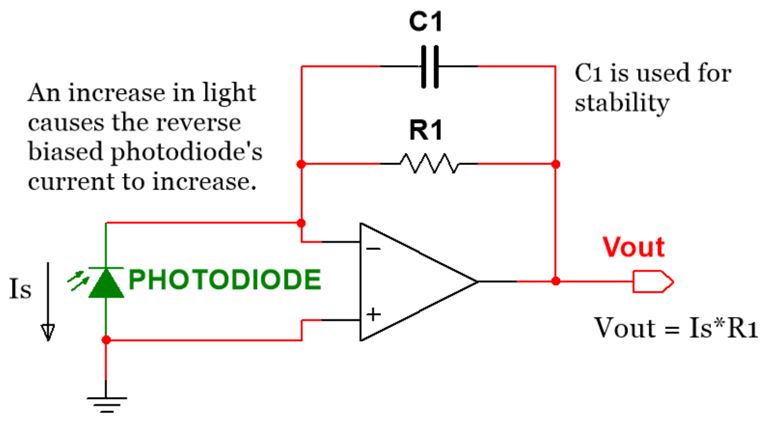
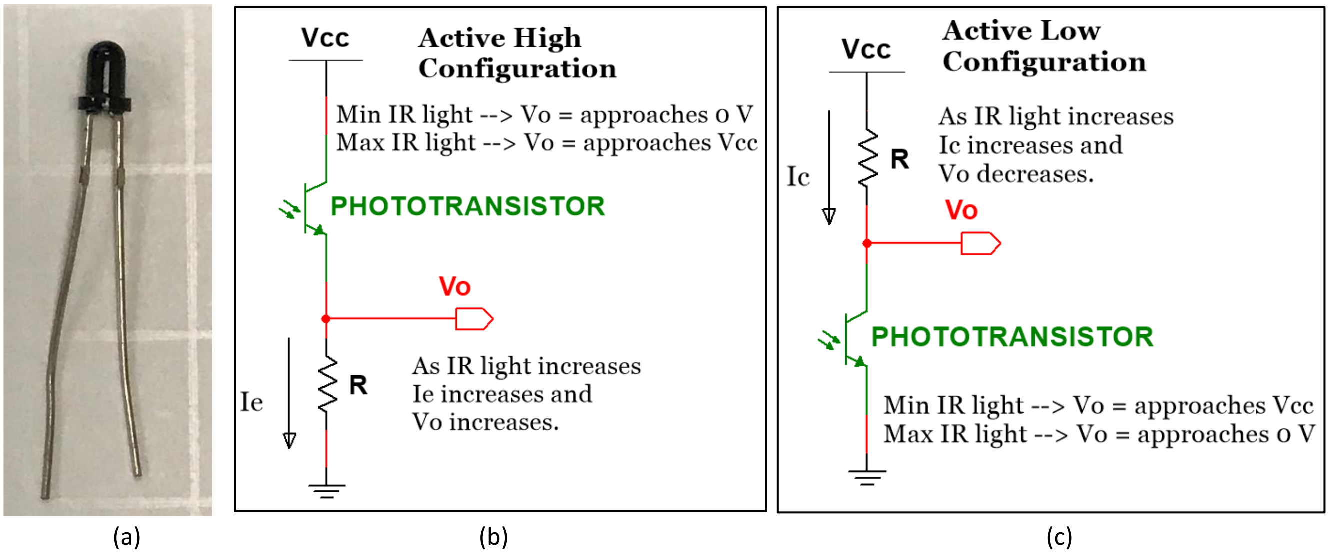

C1
C1 is used for stability
R1
An increase in light causes the reverse biased photodiode's current to increase.
Vout
Is
PHOTODIODE
Vout = Is*R1

Figure 4.18: Transimpedance amplifier

Section 4.2.3 – Photovoltaic sensors: Phototransistors

Another method that is used to convert the photo current output of photodiodes to voltage is to use the photodiode as the source of base current in a bipolar junction transistor (BJT). The internally combined photodiode and BJT is called a phototransistor. The BJT allows for the amplification of the small photodiode current. Since the photodiode current is significantly amplified, the sensitivity of the phototransistor is much higher than the photodiode. However, the response time is much slower in a phototransistor. The following figure shows how the phototransistor is used to amplify the photodiode current and convert it to a voltage.

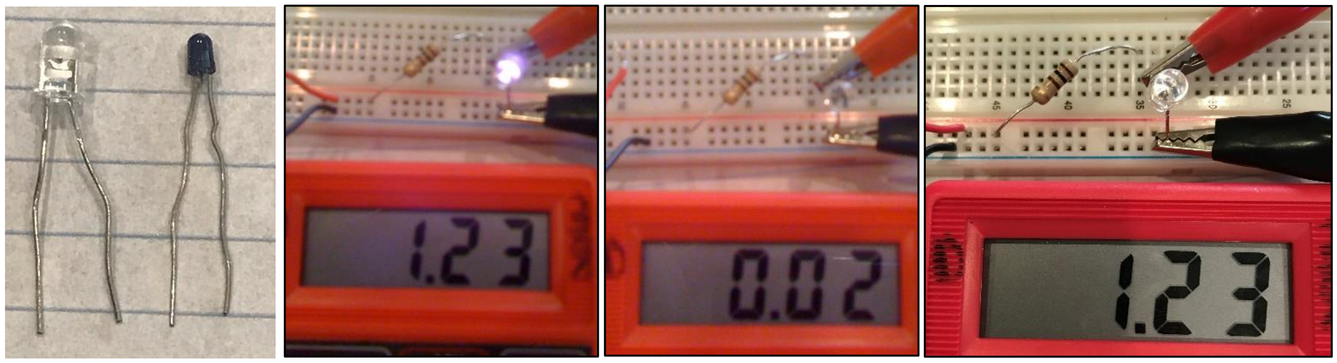
- For the **active high configuration** (Vo↑ as Light ↑) the resistor is connected between ground and the emitter pin of the phototransistor and the output voltage (Vo) is connected to the emitter.
- For the **active low configuration** (Vo↓ as Light ↑) the resistor is connected between Vcc and the collector pin of the phototransistor and the output voltage (Vo) is connected to the collector.


Vcc
Active High Configuration
Min IR light --> Vo = approaches 0 V
Max IR light --> Vo = approaches Vcc
PHOTOTRANSISTOR
Vo
Ie
R
As IR light increases Ie increases and Vo increases.
Vcc
Active Low Configuration
As IR light increases Ic increases and Vo decreases.
Ic
R
Vo
PHOTOTRANSISTOR
Min IR light --> Vo = approaches Vcc
Max IR light --> Vo = approaches 0 V

(a) (b) (c)

Figure 4.19: (a) LTR-4206E Phototransistor (The emitter is the long leg and the collector is the short leg for this type of phototransistor) (Emitter is the arrow side) (b) Active High Configuration, (c) Active Low Configuration

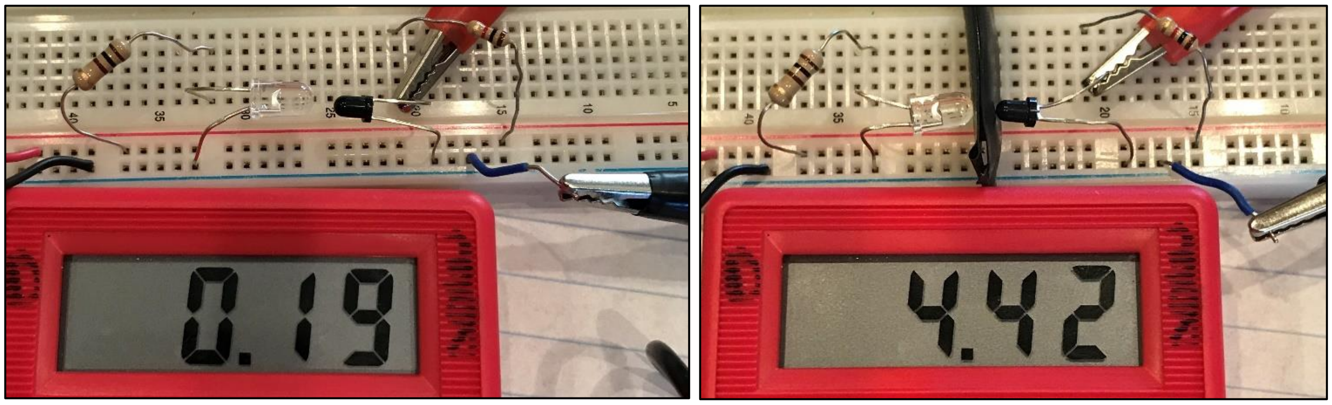
The LTR-4206E phototransistor has a peak sensitivity when detecting IR light at 940 nm. This means it will have the best response (largest current produced) when paired with an infrared (IR) LED that emits light at 940 nm. Figure 4.20a shows photos of two versions of 940 nm IR LEDs (the LTE-3371T has a clear shroud and the TSAL4400 has a dark colored shroud). Figure 4.20 show the LTE-3371T IR LED used in a voltage divider circuit. In Figure 4.20b the IR LED appears to emit a **purple light** (The cover page shows another example of an IR LED) when current flows through the circuit and the voltage across the IR LED is 1.23 V (a typical IR LED voltage). Photo (b) is taken with an **iPhone 3** camera and IR light is not filtered out in that model. IR LED light appears purple when viewed using a device that doesn't filter out IR light, such as the iPhone 3, most front facing phone cameras, or document cameras. Newer iPhone rear-facing cameras filter out IR light, so the IR LED appears off and purple light isn't visible. In Figure 4.20c the red battery clip is unplugged so the IR LED is off. In Figure 4.20d, where the iPhone 7 is used to take the photo, the IR LED appears to be off, but it is actually on (notice the IR LED voltage = 1.23 V). The LED looks off because the IR light is filtered out. The only reason the IR LED in Figure 4.20d looks different than the IR LED in Figure 4.20c is due to it being taken with the higher quality iPhone 7 camera.



(a) (b) (c) (d)

Figure 4.20: (a) LTE-3371T and TSAL4400 940 nm IR LEDs, (b) IR LED on (taken with iPhone 3), (c) IR LED off (red battery wire disconnected), (d) IR LED on, but can't be viewed with the iPhone 7 rear-facing camera.

The IR LED current is equal to (Vs – V_{LED})/R = (4.5-1.23)/100 = 32.7 mA and the power dissipation of the IR LED is equal to 1.23 V * 32.7 mA = 40 mW. This value is well below this IR LED's maximum power dissipation rating of 100 mW [41]. Next, the IR LED is used in a circuit with a LTR-4206E phototransistor. This circuit has an **active low configuration**, so that when IR light is shining directly on the phototransistor the voltage drops to close to 0 V (19 mV) and then when the beam is broken with electrical tape, so that little to no light gets to the phototransistor, the voltage is close to the 4.5 V supply voltage (4.42 V). **Note:** *The phototransistor voltage (V_{CE}) doesn't go to zero when current flows through it because the minimum voltage of $V_{CE,ON}$ is ~ 0.2 V (called $V_{CE,SAT}$).*



(a) (b)

Figure 4.21: Phototransistor in Active Low configuration (a) IR passed, Vo = low, (b) IR blocked, Vo = high.

The next circuit shows how to use a phototransistor as a light controlled current switch. There is no resistor added to limit the green LED current in this circuit, so the current is only limited by the internal resistance of the green LED and the properties of the phototransistor's collector emitter junction. If the voltage source was larger than 4.5 V, the green LED's 20 mA rating could be exceeded if a current limiting resistor wasn't added.

- **Case 1**: Figure 4.22a – When the IR light is shining on the phototransistor, current flows from Vcc through the green LED and phototransistor to ground. The result is the green LED turns brightly on because it is forward biased with an LED voltage of 2.23 V (the LED voltage increases exponentially with current [32]).
- **Case 2**: Figure 4.22b – When the IR light is blocked from the phototransistor, very little current flows from Vcc through the green LED and phototransistor to ground. The result is the green LED doesn't have enough current flowing through it to turn on. The LED voltage is only 1.68 V, which correlates to a near zero current.

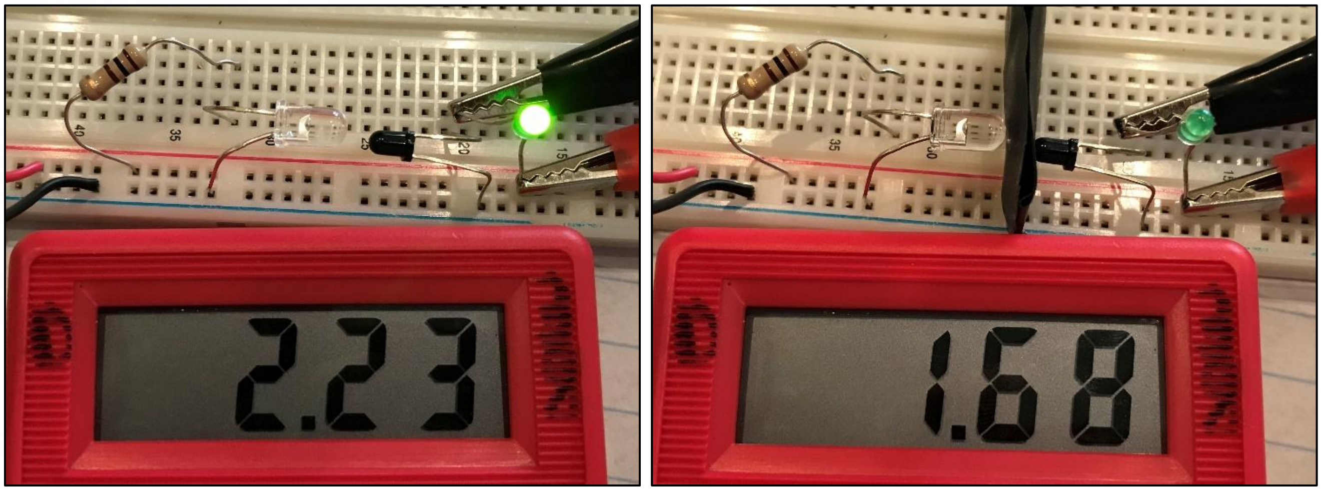
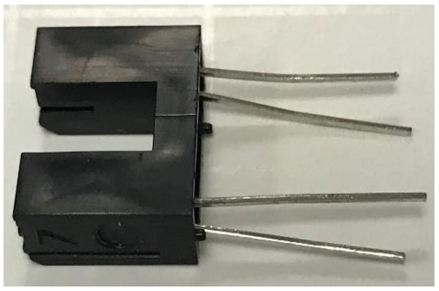


Figure 4.22: Phototransistor circuit with a green LED in current path (a) LED on (b) LED off

The number of applications where phototransistors (or photodiodes) are used is countless. Some of these are shown below with bookmarks to many practical examples that are discussed later in this document.

- Bar Code Scanners
- X-Ray detection systems
- Digital camera control – shutter, autofocus, flash, etc.
- Security systems – When the IR beam is broken an alarm can be triggered.
- Smoke detectors – IR LED and photodiode are put in a chamber and light is reduced with smoke present.
- Ranging sensors (Sharp GP2Y0A21) - Other types of ranging sensors are discussed in Section 4.3.3.
- Optical Encoders - Discussed in more detail in Section 4.3.1.
- The sensor that detects the TV remote control signal – Vishay SOP34838 38 kHz IR Sensor (Figure 4.23)
- Circuits used in card swipe machines that allow you swipe to unlock a door (see photo interrupter).
- Reflection sensor – Act like photo interrupters, but they are angled to reflect off surfaces (Figure 4.24).



The photo interrupter includes the IR LED and phototransistor in one package.

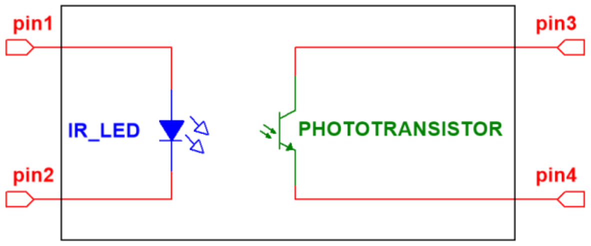


Figure 4.23: Photo of the Sharp GP1S58 Photo interrupter and the block diagram of the internal circuit.

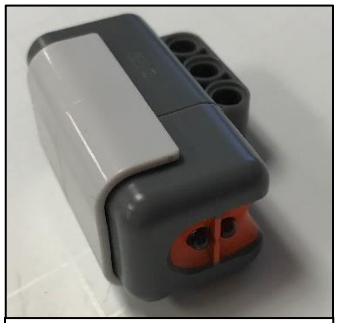


Figure 4.24: Photo of the NXT reflection sensor. The LED and phototransistor look like glass beads.

Reflection sensors use the same circuit as photo interrupters shown in Figure 4.23, but their LED and phototransistor are aimed at precise angles so the light is reflected off a surface that is at a desired distance away. The sensitivity of a reflection sensor will change dramatically as the distance to the surface is changed, so care must be taken on the exact location that it is mounted. Furthermore, bouncing and vibrations must be minimized to avoid spurious readings that occur when the internal phototransistor's height above the surface changes. Figure 4.24 shows the NXT reflection sensor. This sensor is used to distinguish between surfaces that reflect light differently (e.g. a **black surface** will **absorb** most of the light and a white surface will reflect most of the light). IR light is used in most reflection sensors, but the NXT uses red light instead. The reduction in sensitivity (see Figure 4.17) that occurs when using a red LED instead of an IR LED was likely not as important to the designer of this sensor as knowing that the device is actually working. Since this sensor is sometimes used by children, not being able to visibly see IR light and know that it is working was likely deemed to be a customer service nightmare. There is also an NXT reflection color sensor that uses three different colored LEDs to estimate the RGB composition of the color of the surface instead of only the reflectivity %. Both types of NXT reflection sensors can also be used in ambient mode, where the LEDs are turned off and only the ambient lighting is used to trigger the phototransistors.

The following sensor is a Vishay SOP34838 38 kHz IR Sensor. It is used by electronic devices such as TVs to detect the information on a 38 kHz modulated IR signal. It is connected to decoder circuitry or a microprocessor to determine what buttons on the TV remote was pressed. Without decoding, it can be used to drive other circuits when any 38 kHz IR signal is detected. When the 38 kHz IR signal is present the output will drop from Vcc to 0V.

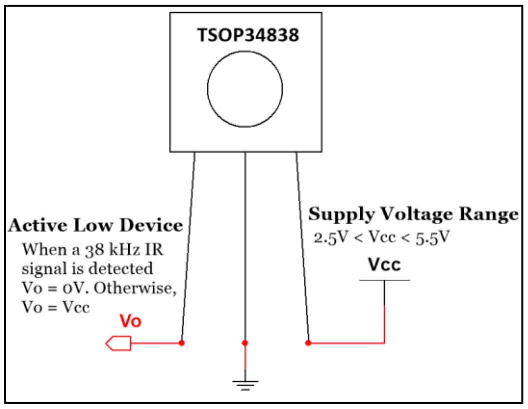


Figure 4.25: Photo & diagram of a Vishay SOP34838 38 kHz IR Sensor

Section 4.2.4 – Photovoltaic sensors: Solar Cells

Solar cells are large area photo diodes that operate in photovoltaic mode, where they have their current restricted so that voltage increases. They are made with semiconductor materials that are optimized to convert visible light to power efficiently [45].

On a small scale, solar cells can be used in consumer electronics for a light controlled voltage source that can be used to charge batteries in mobile devices. The price of these parts depends primarily on their output power, which is closely related to their surface area. The following table shows a list of small solar cells that were available at the

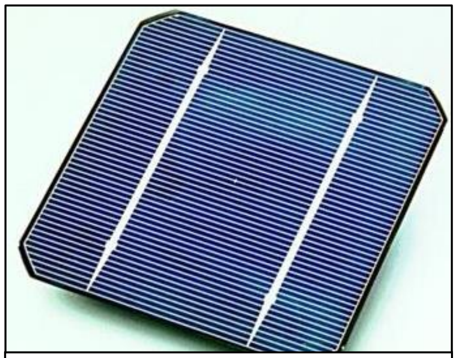


Figure 4.26: Solar Cell. Public Domain: https://en.wikipedia.org/wiki/Solar_cell#/media/File:Solar_cell.png

time of publication. In order to better correlate price to output power, all of these solar cells selected in the table below are from the same manufacturer (IXYS) and from the same product series (IXOLAR). The open circuit voltage (i.e. the maximum voltage the solar cell can output) is also kept fairly close to the same values to achieve a better comparison.

Table 4.4) A list of solar cells available from Digikey. Source: https://www.digikey.com/products/en/sensors-transducers/solar-cells/514?k=solar

Part #	Dimensions	Power	Open Circuit Voltage	Cost	Cost/mW
SLMD48HO412L	35mm X 22mm	109 mW	7.56 V	$7.64	$0.070
SLMD121H10L	42mm X 35mm	223 mW	6.3 V	$13.13	$0.059
SLMD481H08L	89mm X 55mm	714 mW	5.04	$33.5	$0.047
SLMD481H12L	90mm X 79mm	1080 mW	7.56 V	$42.95	$0.040

In larger scale systems, multiple solar cells are combined to create solar panels and PV systems, as shown below.

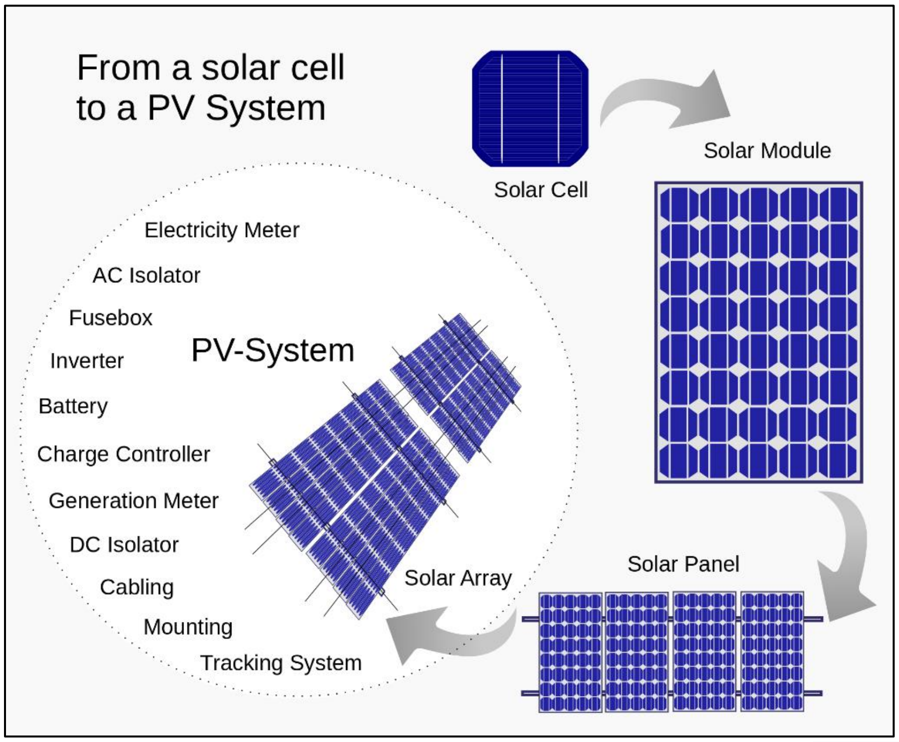


Figure 4.27: Using solar cells to create solar panels and PV-systems. Public Domain: **https://en.wikipedia.org/wiki/Solar_panel#/media/File:From_a_solar_cell_to_a_PV_system.svg**

Section 4.3 – Position Sensors

This section will focus on sensors used to measure rotation, range (or distance), and angular or linear displacement, as summarized in Table 4.5. The accelerometer is another position related sensor that won't be covered in this section, but is important for vibration and acceleration measurements. The following link shows how to make accelerometer measurements in LabVIEW with a DAQ: http://www.ni.com/tutorial/7110/en/

Table 4.5) Position Sensors - Electrical output type and summary.

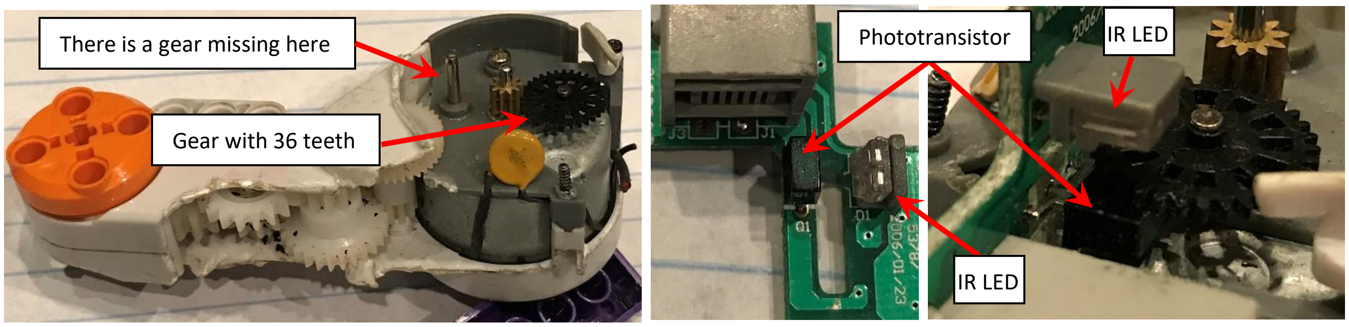
Sensor Type	input/output	Summary
Rotary Encoder (Optical or Magnetic)	angle or rpm /voltage waveform	**Optical encoders** detect the rotation change of a shaft by shining a light through gear teeth or an encoder wheel. When the light is blocked the light sensor outputs a different voltage than when light passes through so that a periodic waveform is produced as the motor shaft rotates. The resulting square wave can be used to determine angle of movement or rpm. **Magnetic encoders** operate in the same fashion, but use magnetic (usually Hall Effect) sensors instead of light sensors.
Linear Variable Differential Transducer (LVDT)	linear displacement /voltage	In an **LVDT**, a transformer core moves with displacement and the output voltage changes proportionally with the relative position of the core. They have a range of up to several inches with an accuracy on the order of micrometers. The sensitivity, k, is used to find displacement from voltage: **Δdisplacement = k · ΔVout**
Range Sensors (Infrared, Laser Ultrasonic)	Distance /voltage pulses	**IR range finding** involves sending out IR light and receiving a reflection with a phototransistor. It is the cheapest and easiest, but is limited to very short ranges and accuracy is not as good because the voltage versus distance is a nonlinear function. **Ultrasonic range finding** has wider ranges, but is more expensive and complicated to implement. **Laser range finding** allows for much larger distances than IR and ultrasonic, but is not good for precision and are usually limited to handheld devices.

Section 4.3.1 – Encoders

Encoders are devices that convert angular position or motion of a shaft or axle to an analog or digital code. There are two types: optical and magnetic. The optical type are used in the Lego Mindstorm NXT motors as shown in the motor that is taken apart in Figure 4.28. An IR LED with a clear shroud shines a light through a black gear with 36 teeth. Assuming that the phototransistor is in active high configuration the following occurs:

- When IR light shines through the openings between the gear teeth, Vo from the phototransistor = ~ Vcc.
- When IR light is blocked by the gear teeth, Vo from the phototransistor = ~ 0V because no light is present.

In this way, the phototransistor in the encoder produces a square wave voltage as the motor shaft (with the gear attached to it) rotates. The square wave can be used to measure the angular velocity in radians per second (rad/sec) or revolutions per minute (rpm) or the number of pulses in the square wave can be counted to determine the amount of angular displacement. Since there are 36 teeth and 36 openings (or slits) between the teeth, the resolution is $360^o/36 = 10^o$/step (which is much worse than most encoders). For example, if 12 pulses are counted from the NXT's phototransistor the angular displacement is $120^o \pm 5^o$.



(a) (b) (c)

Figure 4.28: (a) NXT Motor internal view, (b) Encoder PCB, (c) Encoder sending IR light through gear.

Encoder discs are another option that can be used to block and pass the IR light instead of gear teeth. By adding a wheel with clear and opaque slits a very fine resolution can be attained.

If a magnetic encoder is used, a Hall Effect sensor (a sensor that detects magnetic field strength) is usually used as a "magnetic pickup" to detect metal gear teeth or one or more magnets that rotate with the shaft. This tutorial (http://zone.ni.com/devzone/cda/tut/p/id/4500) shows a figure that illustrates this process. A Hall Effect sensor is shown in Figure 4.29. The SS400 series Hall Effect sensor is a "latching" **digital sensor**.

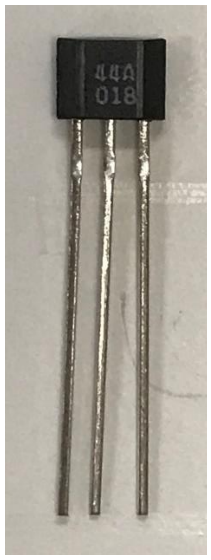
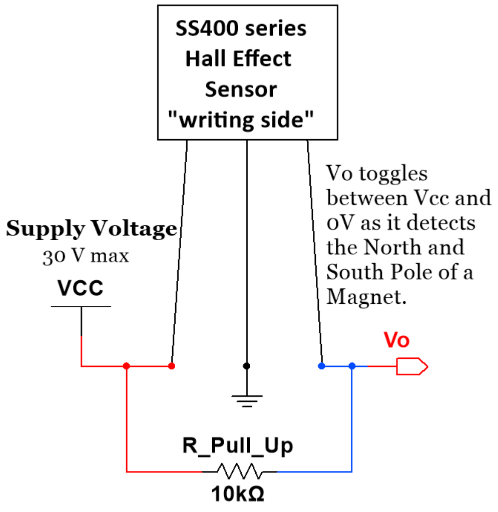


SS400 series Hall Effect Sensor "writing side"
Supply Voltage
30 V max
VCC
Vo toggles between Vcc and 0V as it detects the North and South Pole of a Magnet.
Vo
R_Pull_Up
10kΩ

Figure 4.29: SS400 Series latching Hall Effect digital sensor (See Appendix A provides shows how to measure this sensor with a DAQ).

Since this is a digital sensor it can be sent directly into a digital input line of a DAQ and the ADC process can be skipped as long as Vcc is set to the proper voltage level. Recall: The digital Vcc voltage level is TTL (or 5V) for the myDaq. This sensor acts as follows:

- When the south pole of a magnet is in range of the sensor, its output is 0V
- When the north pole of the magnet is in range of the sensor, its output is 5V.

In this way, as a magnet spins a square wave is produced that can be used to determine angular position or velocity just like as previously described with the optical encoder. The sensor used in Figure 4.26 requires a "**Pull Up resistor**" so that the output is never floating and is forced to Vcc when nothing is connected to the output pin.

Appendix A, provides a detailed step by step guide on how to use LabVIEW with a myDaq to take measurements of a SS400 series Hall Effect sensor and includes all 6 types of DAQ inputs and outputs (analog, digital, counter). Which were discussed in Section 3.2.

The following link contains a document titled "Encoder Measurements: How-To Guide" that provides an overview of encoders and how to make encoder measurements with a DAQ: http://www.ni.com/tutorial/7109/en/

Section 4.3.2 – Linear Variable Displacement Transducers (LVDT)

In an LVDT, a transformer core moves with displacement and the output voltage changes proportionally with the relative position of the core to the coils, as shown in Figure 4.30. They can be used to measure displacement with a range of up to several inches with an accuracy on the order of micrometers. A good LVDT application is to have it connected to a cylinder so that the cylinder's piston position is precisely known as it moves in and out with the LVDT. The displacement versus voltage relationship can be approximated by the following linear equation.

[4.4] **Δ displacement = k · ΔVout** (k = sensitivity – common LVDT units are mV/cm)

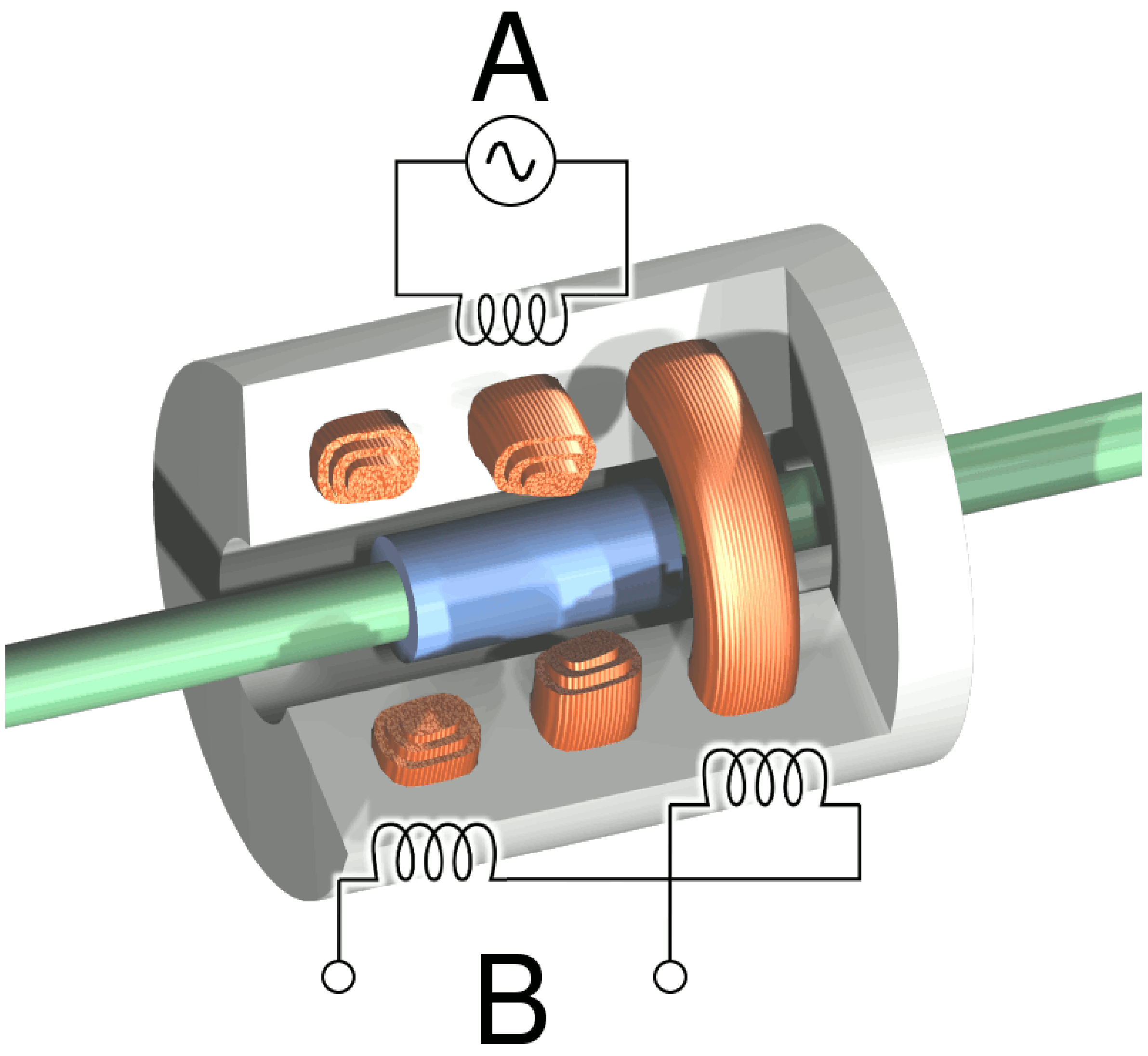


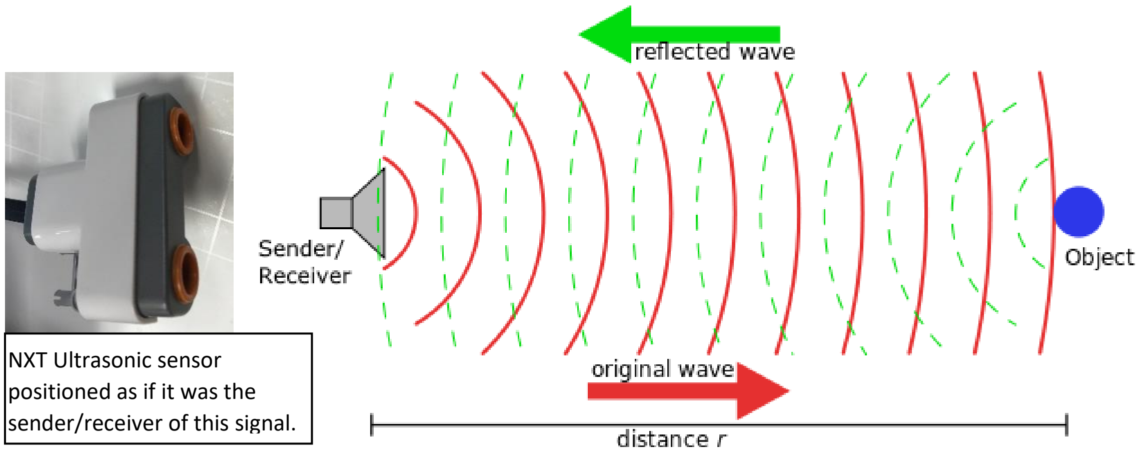
Figure 4.30: Cutaway view of an LVDT. Current is driven through the primary coil (labeled A), which results in the generation of an induction current through the secondary coils (labeled B). © Wapcaplet. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Linear_variable_differential_transformer#/media/File:LVDT.png

Section 4.3.3 – Ranging Sensors

Ranging sensors are used to measure the range (or distance) to an object. There are many ways to implement a ranging sensor. As previously mentioned, an IR LED and a phototransistor can be used as a ranging sensor. **IR range finding** is the most economical form and by far the easiest to implement (you just read the voltage pin that corresponds to distance), but it has a limited range and is not very accurate due to a nonlinear relationship between range and voltage. For example, the Sharp GP2Y0A21 has a range of only 10 to 80cm. As the previous link shows, short range versions of this Sharp sensor can be used to get the minimum range down to 2 cm.

An **ultrasonic range finder** (also called sonar), like the NXT sensor shown below, sends out a high frequency ultrasonic wave that travels at the speed of sound (340.29 m/s) and waits for an echo, as illustrated in Figure 4.31. The range is calculated using equation 4.5. Since the wave has to travel to the object and back, the speed of sound multiplied by the travel time must be divided by 2, as shown in the following equation.

[4.5] **Range = ½ * speed of sound * travel time** where: speed of sound = 340.29 m/s



NXT Ultrasonic sensor positioned as if it was the sender/receiver of this signal.

Figure 4.31: Left: NXT Ultrasonic sensor, Right: Ultrasonic range finding illustration. © Georg Wiora (Dr. Schorsch). Used under CC BY-SA license: https://commons.wikimedia.org/wiki/File:Sonar_Principle_ES.svg#/media/File:Sonar_Principle_EN.svg

One problem with ultrasonic ranging is that sometimes objects aren't detected because they are **too small** or have a **geometry** that causes the reflected wave to travel away from the sender. **Time delays** can also be an issue, especially with the NXT sensor, so if the processing time is too slow then it won't work well. Ultrasound is also much more complicated to implement than IR and typically is done with a microcontroller (see Figure 4.32).

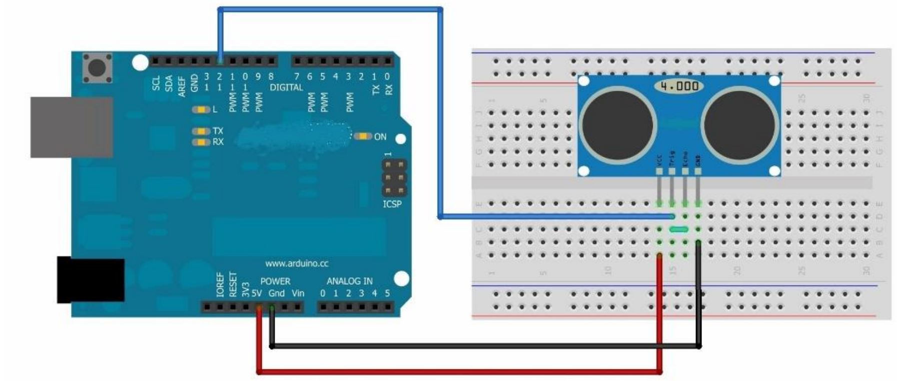


Figure 4.32: Illustration of an ultrasonic range finder connected to an Arduino. © Lahmed2. Used under CC BY-SA license: https://commons.wikimedia.org/wiki/File:Arduino_circuit.jpg

A popular ultrasonic ranging sensor that is frequently used for robotics and is likely the model that was drawn in the illustration in Figure 4.32 is the Parallax Ping (https://www.parallax.com/product/28015). These sensors have a range of 3 cm to 3 m, which is a much wider ranges than is capable with IR sensors. When looking at the side with "eyes" of the parallax ping, the far left pin connects to Vcc and Vcc is required to be 5V. The middle pin is connected to ground and the far right pin is the input/output (I/O) pin. The difficulty in implementing this device is that a precisely timed ultrasonic burst must be sent out and then there must be a precise wait function before it initiates the reading of the echo pulse. The NXT Lego sensor operates in similar fashion.

There are other ultrasonic applications related to range finding that will be discussed after completing this range finder section with a brief overview of **laser range finders**. Like ultrasonic ranging, the equation to determine range is simply the travel time multiplied by the speed of the wave and dividing the result by 2 to account for the send and receive time. Equation 4.6 shows that the only difference for the laser range finding equation is the speed of light is much faster than the speed of sound. This faster speed allows for the measurement of much longer ranges, but it prevents them from being able to get precise sub-millimeter accuracies. For example, some military grade laser range finders like the one shown in Figure 4.33 can measure ranges over 20 km, which is at a completely different level than IR or ultrasonic range finders https://en.wikipedia.org/wiki/Laser_rangefinder.

[4.6] **Range = ½ * speed of light * travel time** where: speed of light = 299,792,458 m/s

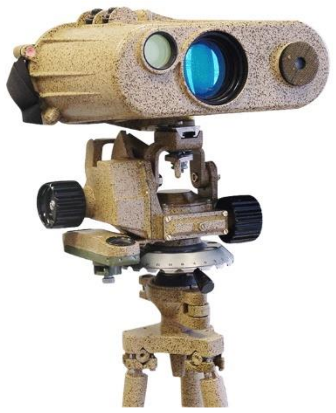


Figure 4.33: Laser Range Finder. Public Domain: https://en.wikipedia.org/wiki/Laser_rangefinder#/media/File:Military_Laser_rangefinder_LRB20000.jpg

Ultrasonic range finding is not only good for detecting distance to an object with a mobile robot, as would likely be the intention of the setup shown in Figure 4.32, but it also has many other applications that use the same principle as previously discussed. First, the frequency that ultrasound uses and three different categories of ultrasonic usages are shown in Figure 4.34. The lowest ultrasonic frequency labeled as "animals and chemistry" likely refers to the approximate frequency at which animals such as bats and dolphins use echo location and how ultrasound is used to affect the chemistry of metals so that their properties are enhanced. The most useful form of ultrasonic testing comes in the high range that is labeled "diagnostic and NDE." NDE stands for Non-Destructive Evaluation and can be used to detect flaws or to look for details of objects that are hidden from sight. This can range from using ultrasound to look at unborn babies to searching for a tiny crack inside a piece of metal. Figure 4.35 shows the form of ultrasonic NDE called flaw detection.

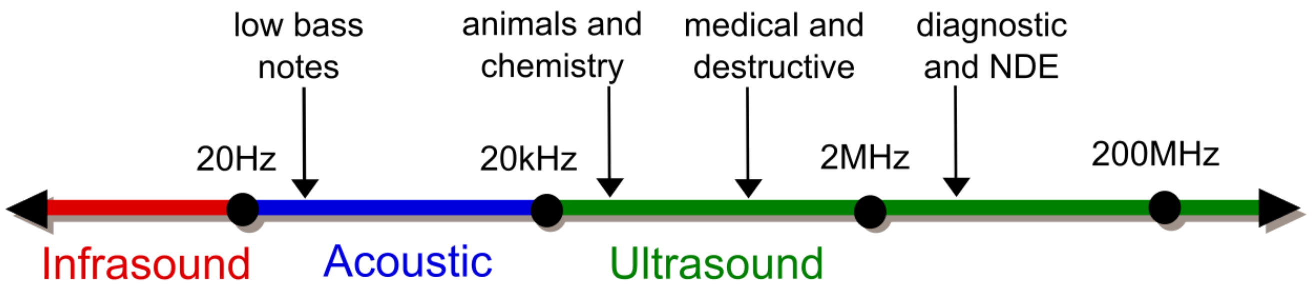


Figure 4.34: Frequency scale. © Coolth. Used under CC BY-SA license: **https://commons.wikimedia.org/wiki/File:Ultrasound_range_diagram.svg**

The figure below shows how an ultrasonic transducer can be used to send out high frequency pulses through a material and wait for the echo. If there are no flaws that have the size or orientation that result in a response, the only reflections (or voltage spikes) will come from the coupling of the transducer to the material and the reflection off of the back wall of the material (as shown in the left waveform in Figure 4.35). If one or more flaws are present, there will be additional voltage spikes. By detecting flaws in materials in this non-destructive manner, problems can be easily detected that could result in a safety issue, such as a gas pipe that has a flaw that will likely result in failure when pressurized. For more information about using ultrasonic for flaw detection see the following link: http://zone.ni.com/devzone/cda/tut/p/id/3368

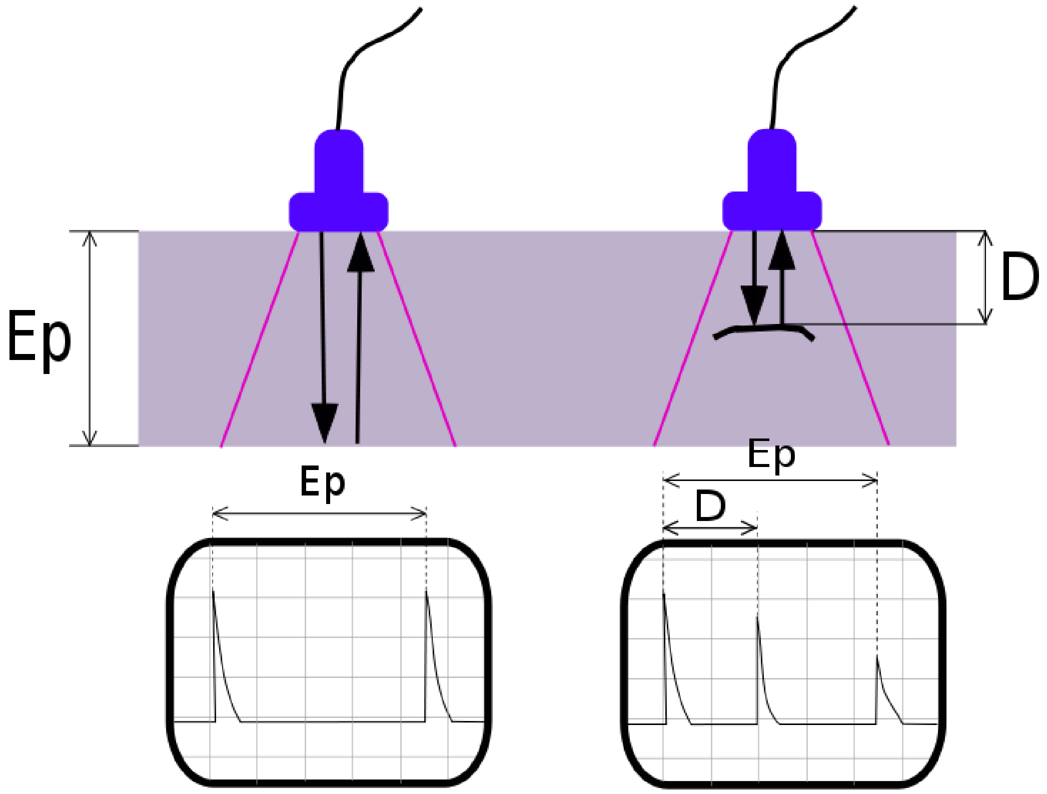


Figure 4.35: Ultrasonic flaw detection. © Romary. Used under CC BY-SA: https://commons.wikimedia.org/wiki/File:UT_principe.svg

Section 4.4 – Load or Strain Sensors

The following table shows the sensors that will be described in this section.

Table 4.4) Load and Strain Sensors - Electrical output type and summary.

Sensor Type	input/output	Summary
Strain Gauge	strain/ resistance	Uses the equation R = ρL/A to detect strain. When an object undergoes tensile strain, the length (and thus resistance) of the strain gauge increases. When compressive strain occurs the length and resistance decrease. A strain gauge has a gauge factor that is used to convert resistance to strain. GF = (ΔR/R)/strain
Strain Gauge Load Cell	static load/ resistance	Load cells measure static forces by using two strain gauges in a half bridge configuration or two pairs of strain gauges (4 total) in a full bridge configuration. In contrast to the standard strain gauge bridge circuit, the load cell contains additional resistors that are used for temperature compensation.
Piezoelectric sensor	dynamic load/Voltage	A dynamic force or pressure squeezes a piezoelectric crystal that creates an electric charge. They have low output voltage so a lot of amplification is needed. They are used for many things such as impact, shock, acceleration, pressure, and vibration.

Section 4.4.1 – Strain Gauges

A strain gauge (shown in Figure 4.36) is made of a pattern of metal wire that, when stressed, changes resistance. The resistance change is directly proportional to the length (L) in the following equation:

[4.7] **R = ρ·L/A**

Where: ρ = resistivity, L = length, and A = area.

As previously discussed in Section 3.4.1, the Whetstone bridge is used to determine the resistance of the strain gauge in one of four methods: 1) Manual method, 2) Quarter Bridge, 3) Half Bridge, or Full Bridge.

Once the resistance of the strain gauge is known the gauge factor of the strain gauge is used to convert resistance to strain [57].

[4.8] **Gage Factor (GF) = (ΔR/R)/(ΔL/L)**

Where:

- ΔR = Change in strain gauge resistance
- R = Initial or unloaded strain gauge resistance
- ΔL = Change in length (if tension ΔL >0, if compression ΔL<0)
- L = Initial length of unloaded strain gauge

[4.9] **Strain = ΔL/L**

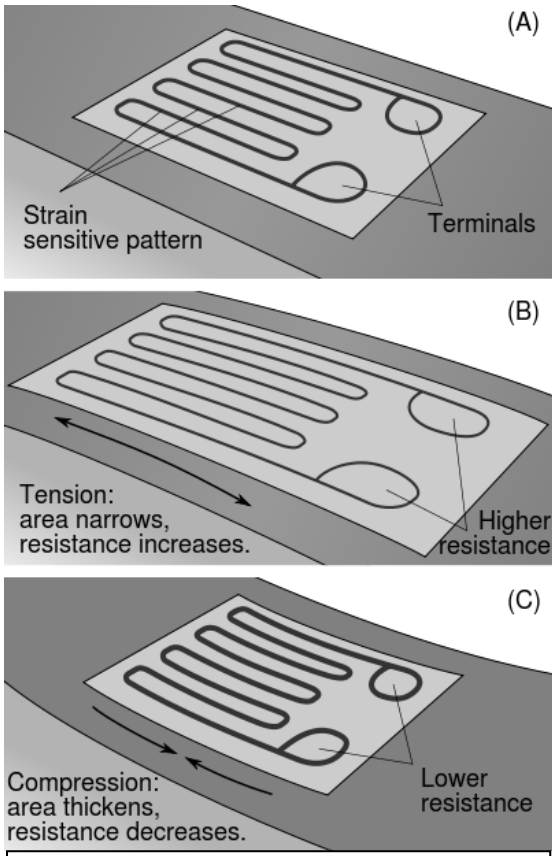

(A)
Strain sensitive pattern
Terminals
(B)
Tension: area narrows, resistance increases.
Higher resistance
(C)
Compression: area thickens, resistance decreases.
Lower resistance

Figure 4.36) Strain Gauges. Public Domain: **https://en.wikipedia.org/wiki/File:StrainGaugeVisualization.svg**

- Metal strain gauges have a gauge factor (GF) of a little over 2 [56]. The GF and measured resistance can also be used to solve for strain → **Strain = (ΔR/R)/GF**.

The following link shows how to connect strain gauges in a full bridge configuration to measure strain in the following situations: tension on a bar, a bending beam, or on a twisted shaft: [http://eln.teilam.gr/sites/default/files/Wheatstone%20bridge.pdf]

Example 4.4) *A metal strain gauge with an unloaded resistance of 1 kΩ was put into a Wheatstone bridge circuit along with two 1 kΩ fixed resistors and a precision potentiometer. For the circuit in Figure 3.9a, R1 and R3 were set as 1 kΩ fixed resistors and R2 was set as the precision potentiometer and when the potentiometer was adjusted to 1.02 kΩ, the V_{DB} output voltage in the bridge was equal to 0 V. If Rx is the loaded strain gauge, calculate the resistance and strain of the strain gauge and determine whether it was a tension or a compression load.*

Based on the description of the circuit the manual method is being used to determine the sensor resistance, Rx.

Since Rx = R2 when R1 = R3 for a balance bridge, the result is **Rx = 1.02 kΩ**

To determine the strain, a GF approximation of 2 will be used since it is a metal strain gauge.

From equation 4.8 $\rightarrow$ GF = 2 = (ΔR/R)/(ΔL/L) = (ΔR/R)/strain $\rightarrow$ Strain = (ΔR/R)/GF = ((1.02-1)/1)/2 = **0.01 m/m**

Since the ΔR was > 0, ΔL must also be > 0. Therefore, the strain gauge was under **tension**.

Example 4.5) *A strain gauge with a gauge factor of 3 and an unloaded resistance of 90 Ω was place in the Rx location in the Wheatstone bridge circuit in Figure 3.9a with a 5 V power supply. If an identical strain gauge was placed in the R3 position and two 50 Ω fixed resistors were used for R1 and R2, what is the value of Rx if the V_{DB} output voltage in the bridge was equal to 1 mV after a load was applied on strain gauge Rx? Also, calculate the strain of the strain gauge in units of µm/m and determine whether it was a tension or a compression load.*

From equation 3.5 $\rightarrow$ **Rx** = [50·90 - (.001 /5)·(50·90+50·90)]/[50 + (.001 /5)·(50+50)] = **89.92803 Ω**

From equation 4.8 $\rightarrow$ strain = (ΔR/R)/GF = ((89.92803-90)/90)/3 = -.00027 m/m = **-270 µm/m**

Since the ΔR was < 0, ΔL must also be < 0. Therefore, the strain gauge was under **compression**.

Section 4.4.2 – Load Cells

Load cells measure static forces by using two strain gauges in a half bridge configuration or two pairs of strain gauges (4 total) in a full bridge configuration. In contrast to the standard strain gauge bridge circuit, the load cell contains additional resistors that are used for temperature compensation. Since they have multiple strain gauges the sensitivity is increased as compared to a single strain gauge. According to [62], strain gauge style load cells have an accuracy of 0.03 to 0.25%.

Figure 4.37 shows a compression style strain gauge load cell. Other common types of load cells include bending-beam load cells and strain load cells that are used to get more accurate strain measurements [62].

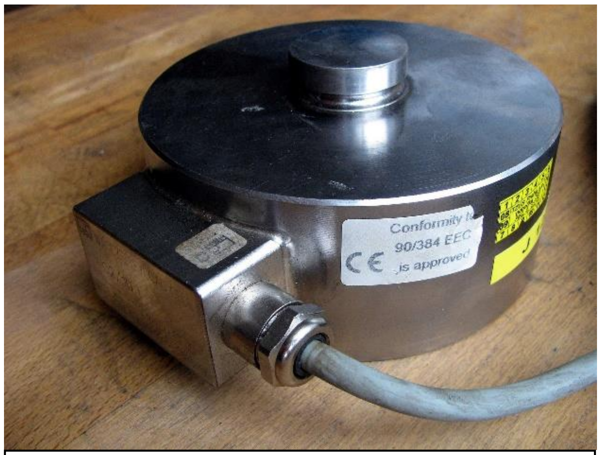


Figure 4.37: Compression style strain gauge load cell. © Jindřich Běťák. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Load_cell#/media/File:Silomer_100kN.jpg

Section 4.4.3 – Piezoelectric Sensors

The piezoelectric effect occurs when a dynamic force or pressure squeezes a piezoelectric crystal that creates an electric charge, as shown in the figure below.

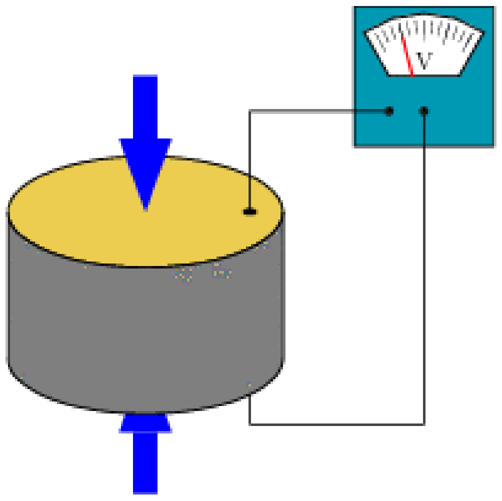


Figure 4.38: This shows a piezoelectric crystal being squeezed and producing a voltage. Click on the link below to see the animation. © Tizeff. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Piezoelectricity#/media/File:SchemaPiezo.gif

Piezoelectric sensors are very rugged and have excellent linearity. Since the devices are linear they will have an equation that uses a sensitivity (k value) to convert voltage to dynamic force (just like the LVDT). One disadvantage is that they have low output voltage, so a lot of amplification is needed. Also, the inability to measure static forces is not ideal because a force that is dynamic for a period of time and then static for a period would need two different types of sensors to measure the force. Piezoelectric sensors are one of the most versatile sensors and are used for a variety of applications such as: impact, shock, acceleration, pressure, and vibration.

The following link provides a white paper that is a guide to explaining how dynamic force sensors work: http://www.pcb.com/contentstore/MktgContent/WhitePapers/WPL_30_Dynamic_Force_Sensors.pdf

There are many other technical white papers and applications notes at www.pcb.com. https://en.wikipedia.org/wiki/Piezoelectric_sensor also has some useful information.

Module 4 – References and Links

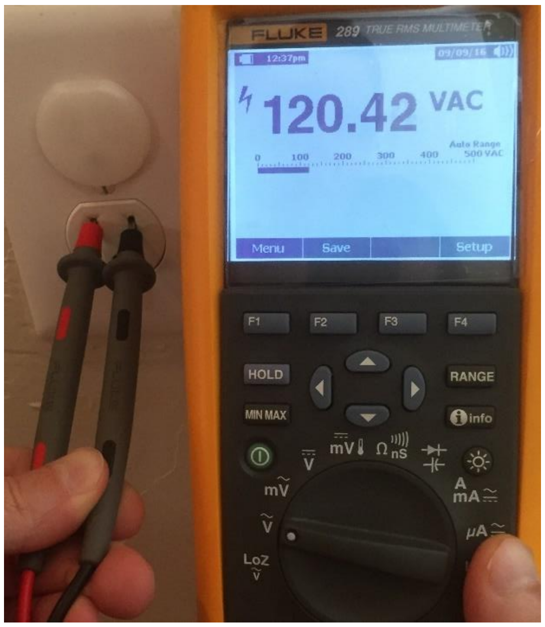
The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[1] Overview of different types of temperature sensors: https://www.allaboutcircuits.com/technical-articles/introduction-temperature-sensors-thermistors-thermocouples-thermometer-ic/
[2] Overview of different types of temperature sensors and active versus passive explanation: https://www.digikey.com/en/articles/techzone/2016/jul/active-versus-passive-temperature-sensors
[3] Type of signal conditioning required for temperature sensors: https://www.omega.com/technical-learning/conditioning-transmission-temperature-sensor-outputs.html.
[4] Details of the physics behind infrared thermometers are shown here: https://www.omega.com/literature/transactions/volume1/thermometers1.html
[5] Details of the physics behind infrared thermometers: https://www.omega.com/literature/transactions/volume1/thermometers1.html
[6] Infrared thermometer Introduction: https://www.omega.com/prodinfo/infraredthermometer.html
[7] Thermocouple information: https://en.wikipedia.org/wiki/Thermocouple
[8] Thermocouple type color codes: https://www.omega.com/techref/colorcodes.html
[9] Thermocouple reference table: https://www.omega.com/prodinfo/thermocouples.html
[10] K-type thermocouple reference chart: https://www.omega.com/temperature/Z/pdf/z204-206.pdf
[11] Thermocouple measurements with a DAQ device in LabVIEW: http://www.ni.com/tutorial/14338/en/
[12] Omega Reference Junction Principles: https://www.omega.com/techref/thermoref.html
[13] Omega Introduction to Thermocouples: https://www.omega.com/prodinfo/thermocouples.html
[14] Omega Introduction to Thermocouple Wire: https://www.omega.com/prodinfo/thermocouplewire.html
[15] Resistance equation from Davis DC Circuits book: https://shareok.org/handle/11244/52245
[16] Callendar-Van Dusen info: https://en.wikipedia.org/wiki/Callendar%E2%80%93Van_Dusen_equation
[17] RTD Specification document: https://www.omega.com/temperature/pdf/rtdspecs_ref.pdf
[18] RTD: https://www.omega.com/toc_asp/frameset.html?book=Temperature&file=RTD_GEN_SPECS_REF
[19] Mouser RTD: http://www.mouser.com/Sensors/Temperature-Sensors/_/N-7gz50?P=1yo26dwZ1yrtlkv
[20] Thin Film RTD elements: https://www.omega.com/pptst/F1500_F2000_F4000.html
[21] NI-DAQ RTD measurement tutorial: (http://www.ni.com/tutorial/14353/en/
[22] 2-wire, 3-wire, and 4-wire RTD configurations: https://en.wikipedia.org/wiki/   thermometer
[23] Datasheet for thermistor in Figure 4.7 and extended Steinhart-Hart equations (see page 4 in datasheet): http://www.mouser.com/ds/2/427/ntcle100-222385.pdf
[24] Steinhart-Hart equations: https://en.wikipedia.org/wiki/Steinhart%E2%80%93Hart_equation
[25] NI-DAQ thermistor measurement tutorial: http://www.ni.com/white-paper/7112/en/
[26] LMT85 datasheet (Analog voltage output, 3-pins): http://www.ti.com/lit/ds/symlink/lmt85.pdf
[27] DS1822+ datasheet (Digital output, 3-pins): http://www.mouser.com/ds/2/256/DS1822-369473.pdf
[28] AD590 datasheet (Analog 1 µA/K current output, 2-pins): http://www.analog.com/media/en/technical-documentation/data-sheets/AD590.pdf
[29] Pn junction overview: https://en.wikipedia.org/wiki/P%E2%80%93n_junction
[30] Electromagnetic spectrum in Figure 4.10: https://en.wikipedia.org/wiki/Light
[31] CdS Cells: https://www.digikey.com/en/product-highlight/a/advanced-photonix/cds-photocells
[32] Overview of LEDs (including wavelength info): https://en.wikipedia.org/wiki/Light-emitting_diode
[33] Photo resistor overview and image in Figure 4.11: https://en.wikipedia.org/wiki/Photoresistor
[34] TSAL 7400 IR LED datasheet: http://www.mouser.com/ds/2/427/tsal7400-205477.pdf

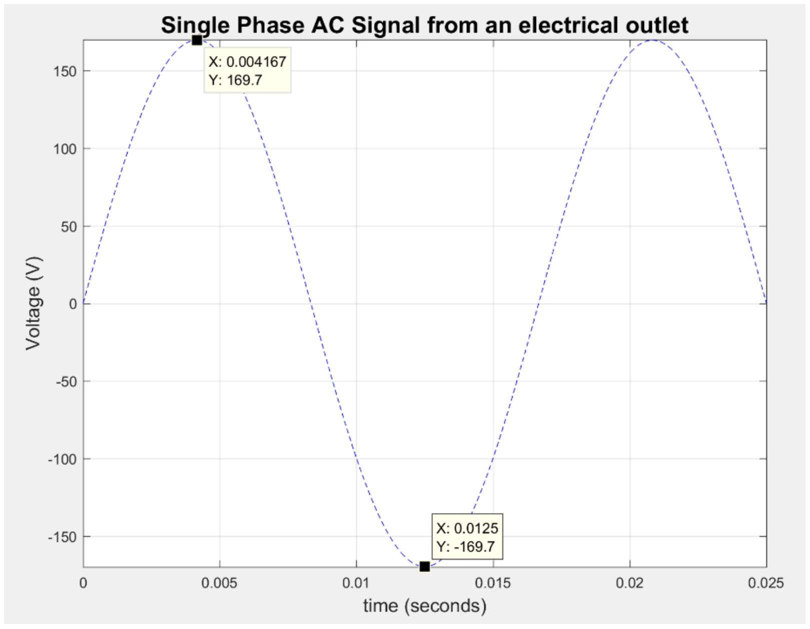
[35] Sharp PD49PI/481PI photodiode datasheet: http://media.digikey.com/PDF/Data%20Sheets/Sharp%20PDFs/PD49PI,%20PD481PI.pdf
[36] Photodiode Overview and image in Figure 4.17: https://en.wikipedia.org/wiki/Photodiode
[37] Overview of transimpedance amplifier design: http://www.ti.com/lit/ug/tidu535/tidu535.pdf
[38] Analog Photodiode Wizard Design Tool video: http://www.analog.com/en/education/education-library/videos/4605084841001.html
[39] Analog Photodiode Wizard Design Tool: http://www.analog.com/designtools/en/photodiode/
[40] LTR-4206E Phototransistor datasheet: http://www.mouser.com/ProductDetail/Lite-On/LTR-4206E/?qs=sGAEpiMZZMtk5EqIReXPtZy9wHdA2YBMQbzLoEBU6nM%3d
[41] LTE-3371T IR LED datasheet: http://www.mouser.com/Search/ProductDetail.aspx?R=LTE-3371Tvirtualkey57820000virtualkey859-LTE-3371T
[42] LTR-4206E phototransistor datasheet: http://www.mouser.com/ProductDetail/Lite-On/LTR-4206E/?qs=sGAEpiMZZMtk5EqIReXPtZy9wHdA2YBMQbzLoEBU6nM%3d
[43] Sharp GP2Y0A21 IR ranging sensor datasheet: https://www.pololu.com/product/136
[44] Vishay SOP34838 38 kHz IR Sensor datasheet: http://www.mouser.com/ProductDetail/Vishay-Semiconductors/TSOP34838/?qs=sGAEpiMZZMvAL21a%2fDhxMhls0hOpvmKUxNBvshin09E%3d
[45] Solar Cell Overview and image in Figure 4.26: https://en.wikipedia.org/wiki/Solar_cell
[46] Solar cell manufacturer website: http://www.ixys.com/
[47] IXOLAR series solar cells: https://www.digikey.com/catalog/en/partgroup/ixolar/13159
[48] How to make accelerometer measurements in LabVIEW with a DAQ: http://www.ni.com/tutorial/7110/en/
[49] Magnetic Encoder Overview: http://zone.ni.com/devzone/cda/tut/p/id/4500.
[50] How to use make encoder measurements with a DAQ: http://www.ni.com/tutorial/7109/en/
[51] LVDT overview and image in Figure 4.27: https://en.wikipedia.org/wiki/Linear_variable_differential_transformer
[52] Ultrasound overview and images used in Figure 4.31, Figure 4.34, and Figure 4.35: https://en.wikipedia.org/wiki/Ultrasound
[53] Parallax Ping Ultrasonic Rangefinder datasheet: https://www.parallax.com/product/28015
[54] Laser Rangefinder overview and image used in Figure 4.33: https://en.wikipedia.org/wiki/Laser_rangefinder
[55] Flaw detection and NDE Overview: http://zone.ni.com/devzone/cda/tut/p/id/3368
[56] Strain gauge overview and image used in Figure 4.36: https://en.wikipedia.org/wiki/Strain_gauge
[57] Omega Strain Gauge Overview: https://www.omega.com/literature/transactions/volume3/strain.html
[58] Applying the Wheatstone Bridge Circuit, Karl Hoffman, http://eln.teilam.gr/sites/default/files/Wheatstone%20bridge.pdf
[59] An Introduction to Measurements Using Strain Gauges, Karl Hoffman, http://www.kk-group.ru/help/Strain_Gauge_Measurements_Book_2012_01.pdf
[60] Bridge Measurement Systems: http://www.ti.com/lit/ml/slyp163/slyp163.pdf
[61] Strain Gauges and Whetstone bridges: https://www.transducertechniques.com/wheatstone-bridge.aspx
[62] Omega Strain Gauge Load Cell Overview: https://www.omega.com/technical-learning/types-of-load-cells.html
[63] Guide to dynamic force sensors: http://www.pcb.com/contentstore/MktgContent/WhitePapers/WPL_30_Dynamic_Force_Sensors.pdf
[64] Website that has a lot of good technical information about piezoelectric sensors: www.pcb.com.
[65] Piezoelectric sensor overview and image used in Figure 4.38: https://en.wikipedia.org/wiki/Piezoelectric_sensor

Module 5 – Power Systems

Alternating current (AC) and transformers are discussed in detail in the Davis AC Circuits book [1]. Completely studying that AC Circuits book is not required reading to understand this book, but in order to better understand AC motors and generators in Module 6, a brief overview of power systems is needed to provide basic knowledge of AC single phase and three phase power. Figure 5.1a shows a Multimeter in AC mode measuring the RMS (Root Mean Square) voltage of a standard US electrical outlet. In this case, the measurement is slightly higher than 120 Volts RMS (or Vrms). The actual voltage signal that comes out of US electrical outlets is a sinusoid with a frequency of ~ 60 Hz and an amplitude of ~ 169.7 V, which comes from the equation: **120 Vrms · √2 = 169.7**.



(a)



(b)

Figure 5.1: (a) Measuring RMS voltage of an AC Electrical Outlet, (b) 120 Vrms, 60Hz Single Phase AC signal.

AC signals are used to transfer power from one location to another. However, voltages much higher than 120 Vrms are used to transmit power across long distances. Using a higher voltage results in a lower transmission cost due to the following (Refer to Module 1 of the Davis DC Circuits eBook [2] for background information):

- Larger voltage results in a lower current (Recall: P = V · I).
- Lower current results in higher wire gauge wire (**Note:** *Higher AWG wire gauge = smaller wire*).
- Higher wire gauge results in less copper material.
- Less copper material results in less cost.

To lower transmission cost, voltage is transmitted at extremely high voltages (in the range of hundreds of kilovolts). Once, it reaches its destination a transformer is used to reduce the voltage to the level required by the user. While North America standardized on 120 Vrms power, which ranges from 114 to 126 Vrms [4], Europe, and most countries outside of North America, use power that is about twice as high (220 to 240 Vrms).

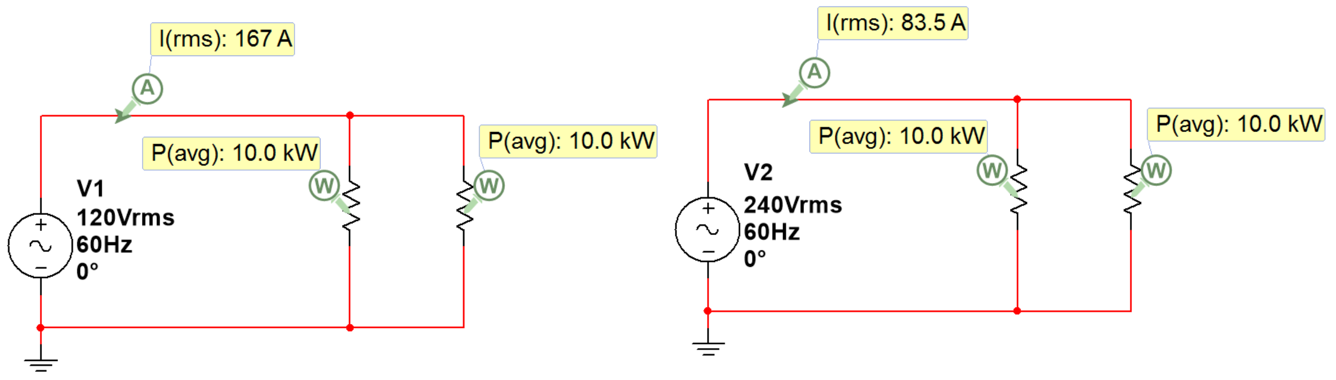
A common way to classify the size of wire is by using the American Wire Gauge (AWG) standard. Table 5.1 shows the diameter of the metal conductor (the plastic shield is not included in the diameter) and typical ampacity ratings for common AWG sizes. Ampacity is a term used to determine how much current the wire can carry before the plastic shield begins to break down. The ampacity rating of a wire depends on the temperature rating of the conductor. For simplicity, Table 5.1 only includes the ampacity of copper at a temperature rating of 60^o C.

Table 5.1) American Standard Wire Gauge (AWG)
Source: https://en.wikipedia.org/wiki/American_wire_gauge

AWG	Diameter (mm)	Copper Ampacity (Amps) at a Temperature Rating of 60^o C
0000 (4/0)	11.684	195
000 (3/0)	10.405	165
00 (2/0)	9.266	145
0 (1/0)	8.251	125
1	7.348	110
2	6.544	95
3	5.827	85
4	5.189	70
6	4.115	55
8	3.264	40
10	2.588	30
12	2.053	20
14	1.628	15
18	1.024	16
20	0.812	11
22	0.644	7
24	0.511	3.5
26	0.405	2.2
28	0.321	1.4

Copper material cost is reduced by increasing the voltage to the 240 V European levels, as shown in Example 5.1.

Example 5.1) *Determine the difference in AWG wire size and amount of copper (according to ampacity ratings in Table 5.1) that is needed to supply power for two 10 kW loads using the 120 Vrms and 240 Vrms standards.*



- **<u>120 Vrms:</u>** Each load has a current equal to I = P/V = 10kW/120 = 83.33 Arms
- **<u>120 Vrms:</u>** Total supply current = 83.33 + 83.33 = ~ 167 Arms (Table 5.1 shows **AWG 0000** is needed).
- The cross sectional area of this wire is (π/4) · (11.684mm)^2 = <u>107.2 mm^2</u>
- **<u>240 Vrms:</u>** Each load has a current equal to I = P/V = 10kW/240 = 41.67 Arms
- **<u>240 Vrms:</u>** Total supply current = 41.67 + 41.67 = ~ 83.5 Arms (Table 5.1 shows **AWG 3** is needed).
- The cross sectional area of this wire is (π/4)·(5.827mm)^2 = <u>26.7 mm^2</u> $\rightarrow$ **Copper is reduced by ~ 75%**

With such significant cost savings, it appears that the higher Vrms European standard is superior to the 120 Vrms North America standard, but the **drawback to using 240 Vrms is safety**. The 120 Vrms standard was adopted because of the reduced likelihood of bodily harm at lower voltages. More information about different types of electricity standards and the history of selecting different voltage levels are shown in reference [4].

Reference [4] also shows an estimation of U.S. residential electricity consumption. It shows that an estimated 50% of the electricity is consumed by space heating/cooling, water heating, refrigeration, and lighting. The remaining half of the electricity is consumed in a variety of different ways. It is interesting to note that dryers (for clothes) was estimated to consume approximately 4% of the residential electricity in the US. Due to the large amount of current that is required in a dryer's heating element to work effectively, most electric dryers in the US operate on a special circuit in the home that runs on 240 Vrms power (**Note:** *240 Vrms power is also frequently referred to as 220 Vrms, just like 120 Vrms power is often referred to as 110 Vrms*). While copper cost is a key factor in building power transmission systems that may span hundreds of miles, the number of kilowatt-hours (kW-hr) that are consumed is the primary concern by residential users since that is what their electricity bill is based on. By increasing the voltage to 240 Vrms the clothes can get dry faster, which results in less kW-hrs used and less money spent on electricity. However, safety is a much bigger concern when dealing with the 240 Vrms circuitry.

Another common type of AC power is **three phase power**, where there are three sinusoids that are used in conjunction to power a device or devices. The phase shift between each signal is 120^o, as shown in Figure 5.2.

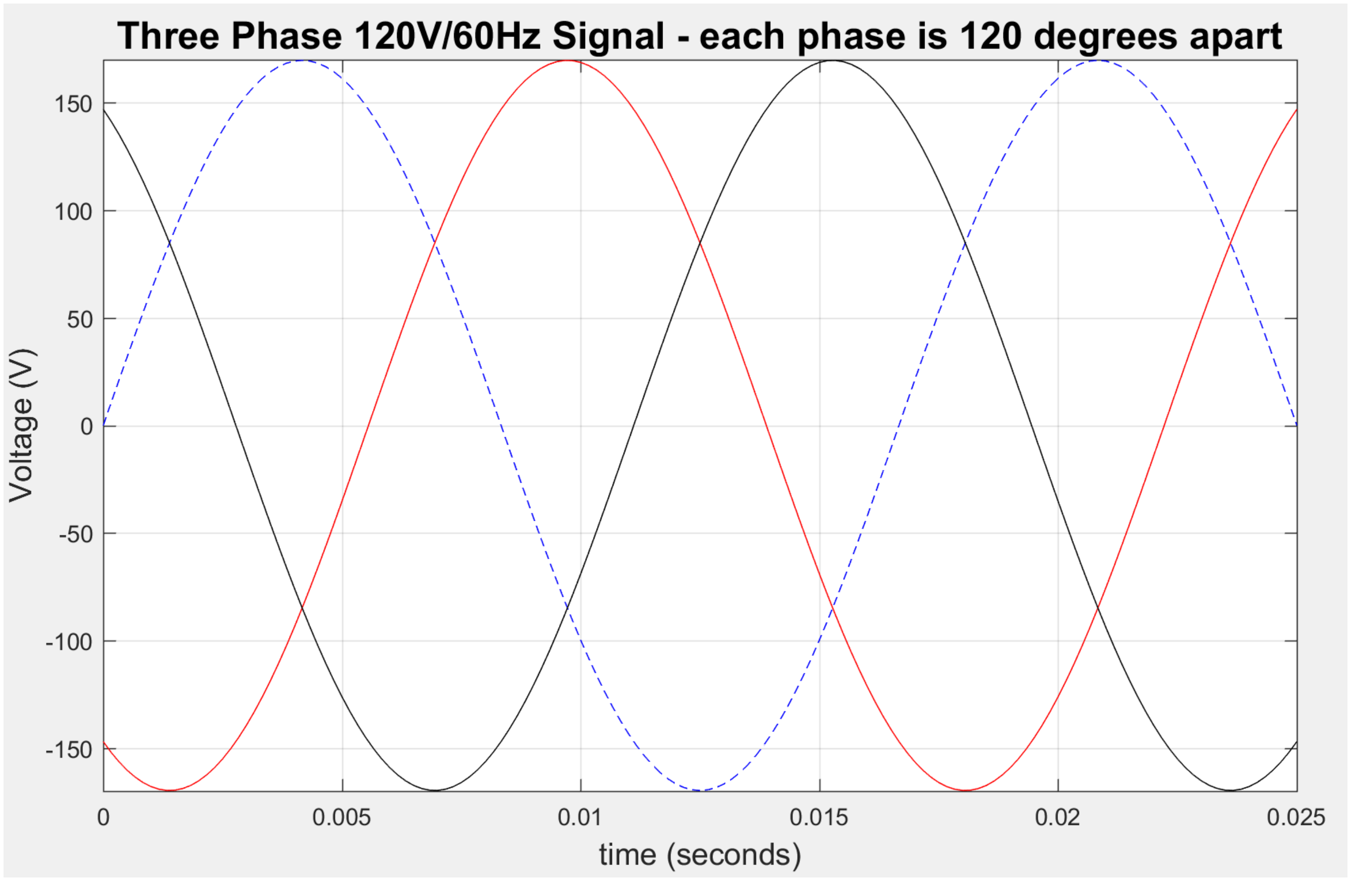


Figure 5.2: 120 Vrms, 60Hz Three Phase AC signal. Blue signal = Vϕ1(t) = 169.7 sin (2π·t+0^o), Red = Vϕ2(t) 169.7 sin (2π·t-120^o), Black = Vϕ3(t) = 169.7 sin (2π·t+120^o). Note: |Vϕ1| = |Vϕ2| = |Vϕ3|

By adding the 3rd wire, **3 times more power can be transmitted versus a 2-wire single-phase system**. When going beyond three phases (e.g. 4-phase) there are diminishing returns. Thus, three phase is the most popular type of polyphaser power system. Three phase power offers the following advantages over single phase power.

- It if more efficient and allows more power to be transmitted with less copper.
- Power is more constant since there are three sinusoids instead of one.
- Voltages can be kept lower for each phase, which allows it to be safer.
- Allows induction motors to starts without special auxiliary windings (discussed more in Module 6).

Three phase is often used for large electrical motors and generators. There are two configurations used with three phase power: Wye (or Y) and Delta. These configurations are shown in Figure 5.3. There are three lines (L1, L2, and L3) that connect to the load and three sources (or phases), that are represented by color rectangles.

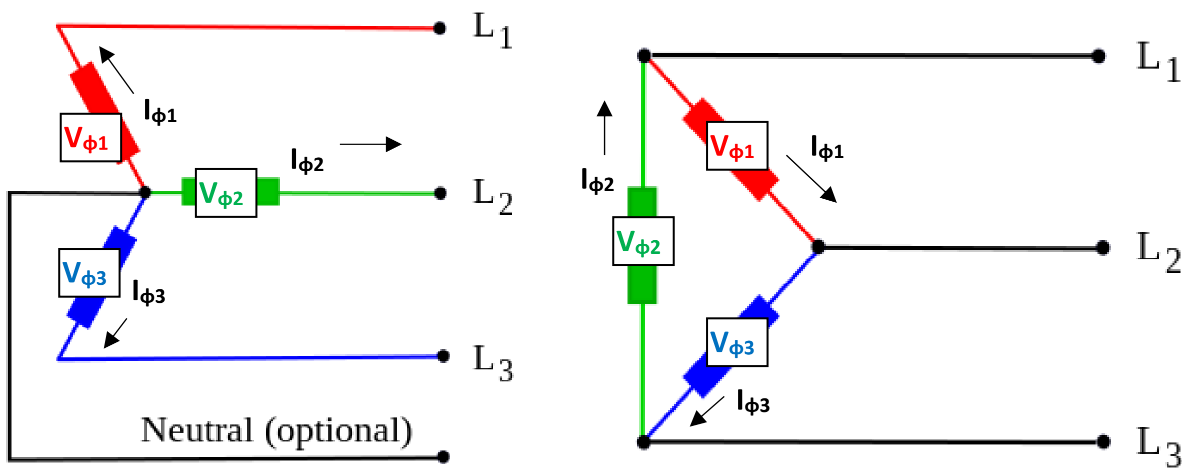


Figure 5.3: (Left) Wye configuration, (Right) Delta configuration. © Svjo. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Three-phase_electric_power#/media/File:The_basic_3-phase_configurations.svg

Note: The magnitudes of the phase voltages are assumed to be the same → **$|V\phi1| = |V\phi2| = |V\phi3|$**

The **Wye** configuration contains 3 lines and a neutral wire. The voltages between each line and the neutral are called the line to neutral voltages ($V_{line\text{-}neutral}$) or phase voltages (V_ϕ) and the voltages between each two sets of lines are called the line to line voltages ($V_{line\text{-}line}$) or line voltages (V_{line}). There are 3 different phase voltages ($V_{\phi1}$, $V_{\phi2}$, $V_{\phi3}$) and there are 3 different line to line voltages ($V_{line1\text{-}line2}$ $V_{line2\text{-}line3}$ $V_{line1\text{-}line3}$). There is only one type of current in a Wye configuration and it is called the line current. Each line has a different line current. For the three lines they are (I_{line1}, I_{line2}, I_{line3}). To convert the magnitude of the phase voltages to the line to line voltages, Equation 5.1 is used (**Note:** *There is also a 30° phase change between the line to line and phase voltages*).

[5.1] **WYE** → $V_{line-line} = V_{line} = \sqrt{3} \cdot V_\phi$ **Note:** *There is only one current for each line* → $I_{line} = I_\phi$

Note: If the neutral wire in Figure 5.3 is ungrounded then Equation 5.1 is only valid if the system is connected to a balanced source and the load impedances are balanced. If the neutral wire in Figure 5.3 is grounded, the loads can be imbalanced and the equation still holds true as long as the source is still balanced.

The **Delta** configuration contains only 3 lines and the currents are different between the phase and line, but the voltages are the same. The currents at each line are called the line currents (I_{line1}, I_{line2}, I_{line3}) and the currents at the sources (i.e. color boxes in the Figure 5.3) are called the phase currents ($I_{\phi1}$, $I_{\phi2}$, $I_{\phi3}$). There is only one type of voltage in a Delta configuration and it is called the phase voltage. Each leg of the Delta configuration has a different phase voltage ($V_{\phi 1}$, $V_{\phi 2}$, $V_{\phi 3}$). To convert the magnitude of the phase currents to the line currents, Equation 5.2 is used (**Note:** *There is also a 30^o phase change between the line and phase currents*).

[5.2] **Delta** → $I_{line} = \sqrt{3} \cdot I_{\phi}$ **Note:** *There is only one voltage for each line* → $V_{line-line} = V_{line} = V_{\phi}$

Example 5.2) *If a WYE configuration was used and the phase voltages are equal to 120 Vrms, what are the line to line voltages equal to?*

Using Equation 5.1, the line voltages are equal to: $V_{line-line} = V_{line} = \sqrt{3} \cdot V_{\phi} = \sqrt{3} \cdot 120\ V = \mathbf{208\ V_{rms}}$

Due to the widespread use of WYE three phase systems with 120 Vrms phase amplitudes, the 208 V is a very commonly used voltage level.

Example 5.3) *If a Delta configuration was used and the phase currents are equal to 2.31 Arms, what are the line currents equal to?*

Using Equation 5.2, the line currents are equal to: $I_{line} = \sqrt{3} \cdot I_{\phi} = \sqrt{3} \cdot 2.31\ A = \mathbf{4\ A_{rms}}$

For more information about three phase power systems, reference [6] is suggested reading. Additionally, reference [7] contains a worksheet that provides a test over some of the basic principles of three phase power that are covered in Chapter 10 of reference [6].

Module 5 – References and Links

The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[1] Davis, *AC Circuits*, 2017, https://shareok.org/handle/11244/51946
[2] Davis, *DC Circuits*, 2016, https://shareok.org/handle/11244/52245
[3] AWG wire gauge: https://en.wikipedia.org/wiki/American_wire_gauge
[4] Information about electricity standards: https://en.wikipedia.org/wiki/Mains_electricity
[5] Three Phase Power Information: https://en.wikipedia.org/wiki/Three-phase_electric_power
[6] More Three Phase Power Information: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-10/three-phase-power-systems/
[7] Three-Phase Power Worksheet: https://www.allaboutcircuits.com/worksheets/delta-and-wye-3-phase-circuits/

Module 6 – Electric Machines

Electrical machines include a wide ranging set of topics and fully covering all of the details would require its own book. This module focuses on practical information about common types of motors and is broken up as:

- Section 6.1 – Electric Machine Overview – Summary of common motor types are included in Table 6.1.
- Section 6.2 – Permanent Magnet DC Motors – These will be referred to as PM-DC Motors.
- Section 6.3 – Wound Field DC Motors – Can be wound as Series, Shunt, or Compound (See Figure 6.9).
- Section 6.4 – AC Motors – Can be designed for single phase power or poly phase (e.g. 3-phase power).

Section 6.1 – Electric Machine Overview

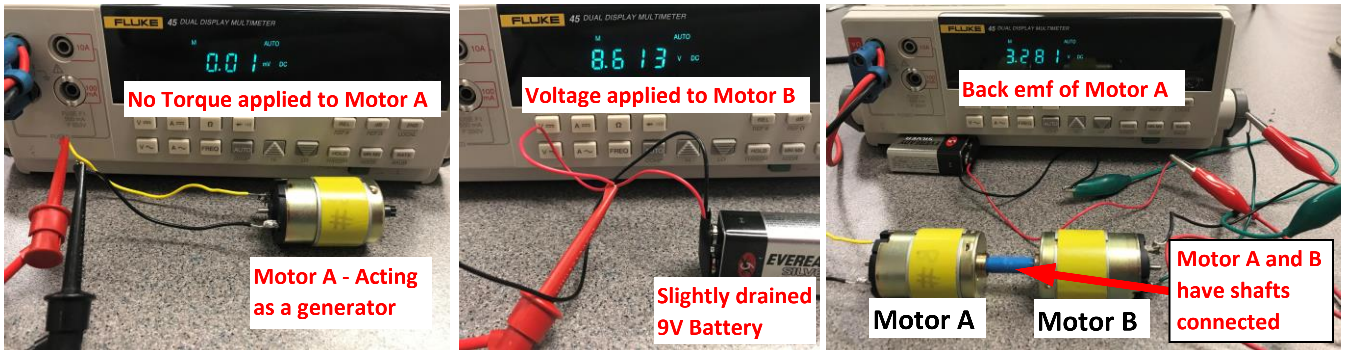
Electrical machines are generally separated into two categories:

- **Generators** – *Convert mechanical power to electrical power & governed by "generator action"* (Equation 6.1).
- **Motors** – *Convert electrical power to mechanical power & governed by "motor action"* (Equation 6.3).

Since this book is focused on electromechanical systems, motors will be emphasized, but many of the principles will also apply to generators. Generator action is essentially the reverse of motor action. Instead of a voltage being applied that results in magnetic field interaction that causes a motor to rotate, the generator has the rotation performed mechanically, which results in a magnetic field interaction that causes a voltage to be produced. There are endless different types of motors and covering them all would be a difficult undertaking. This book will provide a practical overview of some of the more common types that are used and will provide more emphasis on Permanent Magnet DC Motors (PM-DC). A very useful section is included that shows how to do speed versus torque plots and use them to perform calculations for PM-DC motors. Being able to perform these calculations is important because it provides assistance in selecting a motor for a particular application. Electric machines have a rotating part called a rotor and a stationary part that is called a stator. Generators are the primary focus when studying power transmission systems, but it is important to note that most motors also act as generators when their rotors spin. With a motor, applying a voltage causes the rotor to spin, but as the rotor spins a smaller voltage is also produced that works against the applied voltage. The voltage produced as a motor spins is called back electromotive force or back emf. Emf is the basis for how generators work and is governed by Equation 6.1, where **B** is the magnetic flux density in Tesla, **l** is the length of wire in m, and **v** is the velocity that the rotor rotates in m/s. The units of emf are volts [1].

[6.1] $\boldsymbol{emf = B \cdot l \cdot v}$ * This is called the **Generator Action** Formula. It assumes B and v are perpendicular.

Figure 6.1 demonstrates the back emf that is produced by a motor when it is configured as a generator and torque is applied to its shaft. 8.613 V is applied to Motor B causing it to spin. The shaft of Motor B is connected to Motor A, which produces a torque that causes its shaft to spin as if it were a generator. The voltage (back emf) across Motor A is ~ 0V when no torque is applied, but when Motor B is connected to a 9V battery and the two shafts are connected together a back emf of 3.281 Volts is produced (see the far right photo in Figure 6.1).



No Torque applied to Motor A
Motor A - Acting as a generator
Voltage applied to Motor B
Slightly drained 9V Battery
Back emf of Motor A
Motor A
Motor B
Motor A and B have shafts connected

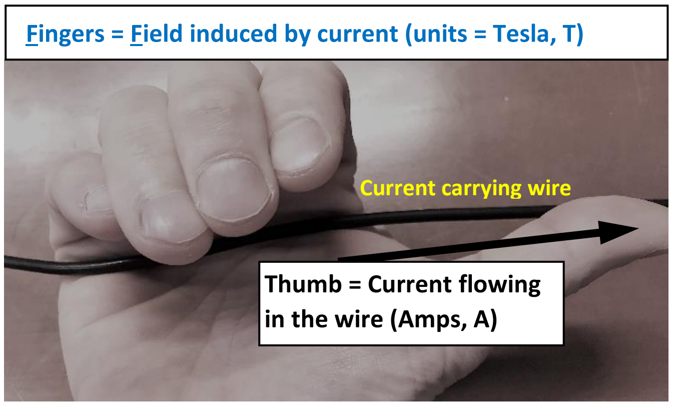
Figure 6.1: Demonstration of back emf. Voltage applied to Motor B → Motor A acting as a generator.

In addition to the back emf that is produced as a result of applying a fixed voltage across a motor there is another electromotive force called counter emf (cemf). This emf (or voltage) is produced when the current in a motor changes over time. Equation 6.2 shows the magnitude of the counter emf. **L** is the inductance and the unit for L is the Henry (H). The units for i and t are Amps and seconds, respectively [1]. This equation is also the voltage that is across an inductor when current changes as described in equation 3.8 in the Davis DC circuits book [13].

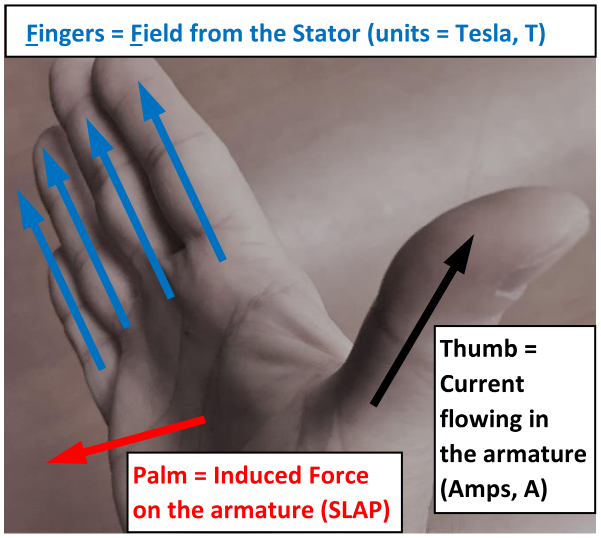
[6.2] $$\boldsymbol{cemf = v_L = L \cdot \frac{di}{dt}}$$ * This equation gives the cemf magnitude. A minus sign is often included.

Caution: *Due to this equation, inductance inside the motor results in a large voltage spike when it is turned on or off. This voltage spike can cause damage to any circuit that it is connected to. In order to prevent this damage a protection diode can be used as describe in Section 6.3 of the Davis AC Circuits book [14].*

To understand how a motor works, the electromagnetism interaction within the motor needs to be discussed. The most fundamental relationship between electricity and magnetism is the **Right Hand Grip Rule** shown in Figure 6.2a. This rule states that if you grip a current carrying wire with your right hand and stick your thumb out in the direction of the current flow, the magnetic field that is induced will wrap around the wire in the same way as your fingers. This direction assumes a Hole Flow model for current (i.e. current flows in the opposite direction of electron flow) [1].


Fingers = Field induced by current (units = Tesla, T)
Current carrying wire
Thumb = Current flowing in the wire (Amps, A)

(a)

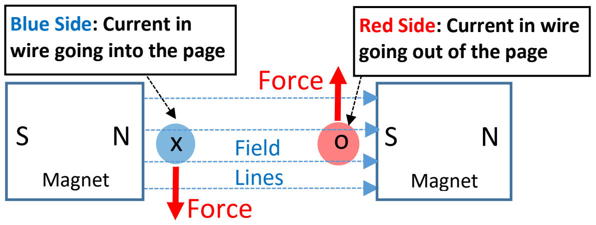

Fingers = Field from the Stator (units = Tesla, T)
Thumb = Current flowing in the armature (Amps, A)
Palm = Induced Force on the armature (SLAP)

(b)

Figure 6.2: (a) Right Hand Grip Rule, (b) Right Hand Palm Rule

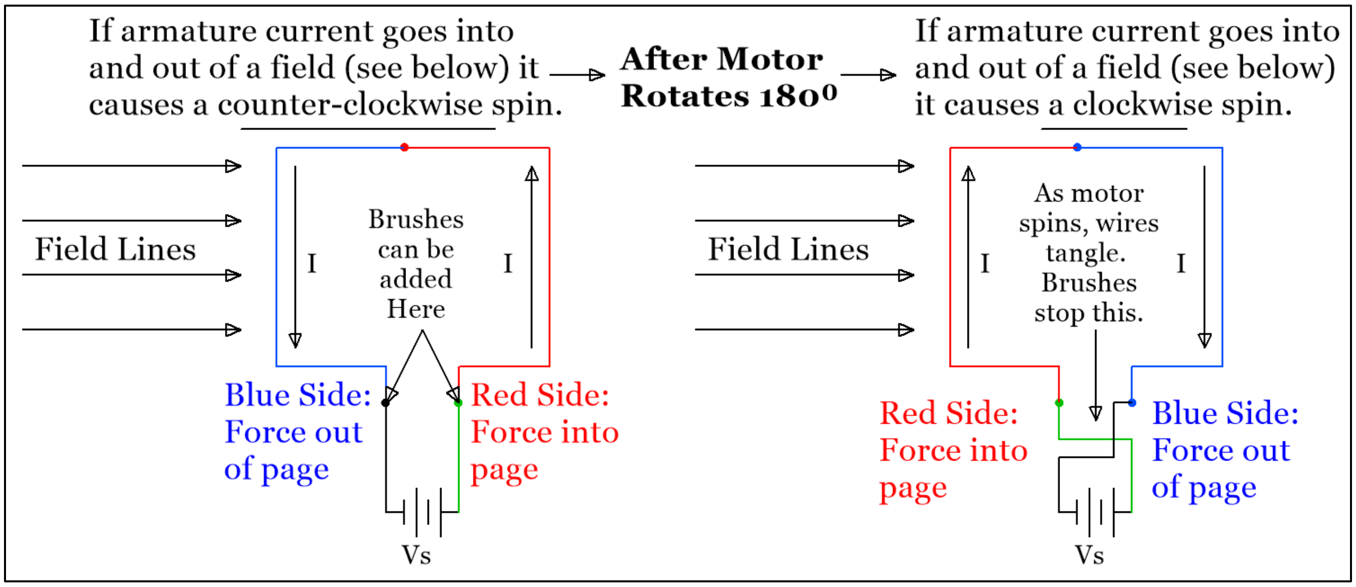
The magnetic field created by current flowing through a wire and how it affects nearby wires is usually neglected when solving circuit problems (as was the case in the Davis DC Circuits book [13]), but it is a source of noise in circuits and a major issue when miniaturizing circuitry. In most motors, a magnetic field is created inside the rotating internal part of the motor (i.e. rotor) by current flowing through an armature. When referring to motors the terms rotor and armature are often used interchangeably since the armature is usually part of the rotor. There is also a magnetic field created by either permanent magnets (see Section 6.3: PM-DC Motors) or an electro-magnet in the stator that interacts with the internal magnetic field created by the armature. The result of this magnetic field interaction is the motor action formula in Equation 6.3. The direction of force is shown in the **Right Hand Palm Rule** shown in Figure 6.2a. To help remember this rule the fingers look like field lines, the thumb looks like an **I** (the symbol for current), and the **force coming out of the palm** of the hand can be remembered by thinking of it as a **slap**.

Figure 6.3 shows an illustration of the motor action formula (Equation 6.3), where current flows into and out of a magnetic field through a wire that represents the armature in a motor. While Figure 6.3 shows the front view of the force on a wire, Figure 6.4 shows the top view where the entire armature wire loop can be seen, along with its connection to a battery.



Blue Side: Current in wire going into the page
Red Side: Current in wire going out of the page
Force
S
N
Magnet
X
Field
Lines
O
S
N
Magnet
Force

Figure 6.3: Motor Action Formula illustration



If armature current goes into and out of a field (see below) it causes a counter-clockwise spin.
After Motor Rotates 180⁰
If armature current goes into and out of a field (see below) it causes a clockwise spin.
Field Lines
I
Brushes can be added Here
I
Blue Side: Force out of page
Red Side: Force into page
Vs
Field Lines
I
As motor spins, wires tangle. Brushes stop this.
I
Red Side: Force into page
Blue Side: Force out of page
Vs

a)

(b)

Figure 6.4: (a) Top View of Figure 6.3, (b) Armature rotated 180^o demonstrating how supply wires tangle and that the force on the armature changes directions and opposes initial rotation (i.e. θ = 270^o in Equation 6.3).

The following link shows a popular YouTube video that shows how to build a simple motor and includes illustrations of the forces on the current carrying wire: https://www.youtube.com/watch?v=it_Z7NdKgmY

Note: *Electron flow (i.e. current flowing out of the negative side of a battery) is shown in this video instead of the more customary Hole flow (i.e. current flows out of the positive side of a battery). When using the electron flow model the force in the Right Hand Palm Rule (*Figure 6.2b*) is reversed and comes out of the back of the hand.*

[6.3] $\boldsymbol{F = I \cdot L \cdot B \cdot \sin\theta}$ * This is called the **Motor Action** Formula

- F = Force in Newtons (N)
- I = Current in the armature in Amps (A)
- L = Length of armature wiring in meters (m)
- B = Magnetic flux density in Tesla (T)
- θ = Angle between wire and magnetic field.

Note: *The maximum force occurs when θ = 90^o (i.e. when the wire is perpendicular to the field).*

- The Right Hand Palm Rule can be used to determine the directions of the forces in Figure 6.3.

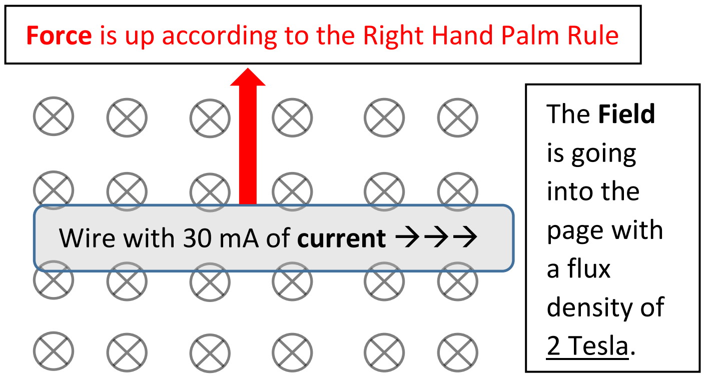
Equation 6.3 can also be written in terms of the positive charge velocity by converting I·L to q·v as shown below:

F = I · L · B · sin(θ) → I = **q/t** → F = **q** · **(1/t)** · **L** · B · sin(θ) → v = **L/t** → **F = q · v · B · sin(θ)** (q=charge in Coulombs)

This equation uses the Hole Flow Model of current flow. If the Electron Flow Model of current flow was used it would give the force due to the negative charge velocity.

Example 6.1) *Determine the direction and magnitude of the Force in the following illustration, where:*

- *The magnetic flux density is equal to 2 Tesla going into the page.*
- *30 mA of Current is moving from left to right using the Hole Flow model for current.*
- *The wire has a length of 7.5 meters.*


Force is up according to the Right Hand Palm Rule
Wire with 30 mA of **current** →→→
The **Field** is going into the page with a flux density of 2 Tesla.

Solution) According to Equation 6.3, the force produced in this situation is:

$$\boldsymbol{F = I \cdot L \cdot B \cdot \sin\theta = 0.03A \cdot 7.5m \cdot 2T \cdot \sin 90^0 = 0.45\ N}, \textbf{direction is Up}$$ (Right Hand Palm Rule is used)

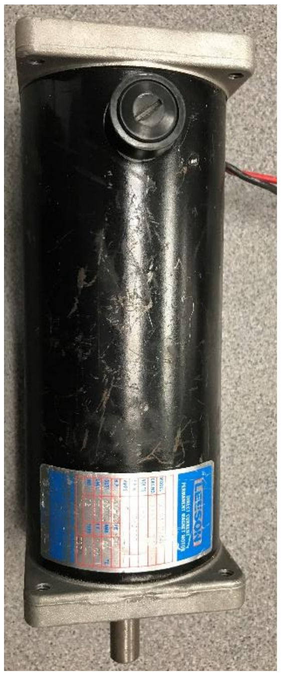
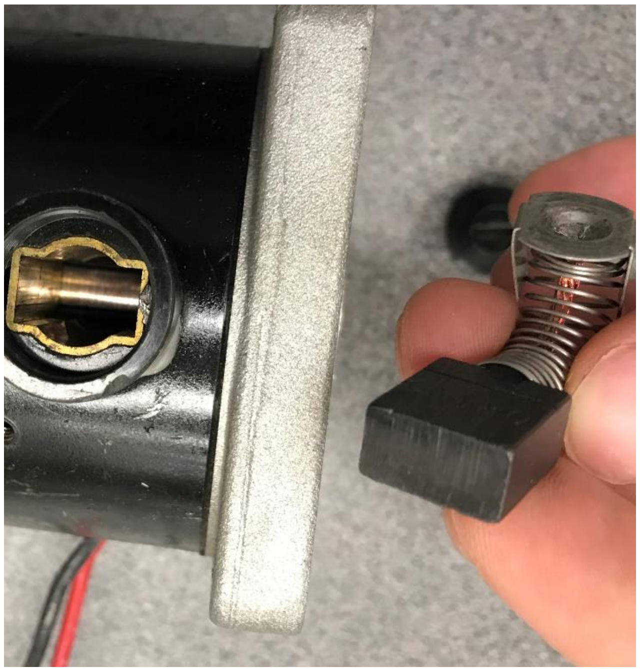
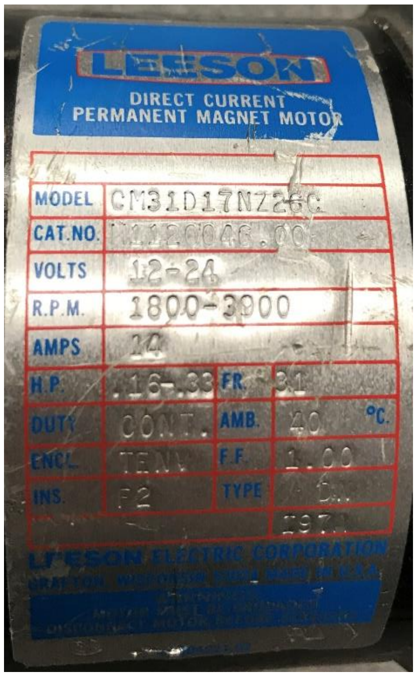
Figure 6.4 highlights **two problems** (explained below) that are **corrected by** adding **brushes** and **commutation**:

1. Supply wires will get tangled up as the armature rotates if **brushes** (see Figure 6.5b) aren't included.
2. The motor action equation (Eq. 6.3) results in rotation that alternates from clockwise to counter-clockwise each time the armature rotates 180^0 without a mechanism to reverse the polarity of the current. **Commutation** (shown in Figure 6.6) is the solution to this problem.

A photo of a Permanent Magnet (PM) DC motor along with one of its spring loaded carbon brushes that allows current to pass from the supply voltage to the armature are shown in Figure 6.5b.

✓ Brushed motors have low initial cost as compared to brushless motors. Brushed motors also have high reliability, but due to brush wear they require a lot of maintenance and have a limited life span [2].

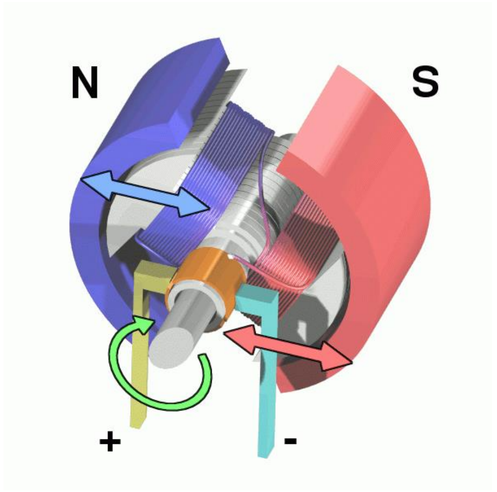
A photo of the nameplate is given in Figure 6.5c to allow the details of this motor to be known and also to highlight the importance of a motor's nameplate. The National Electrical Manufacturing Association (NEMA) produced a motors and generators standards document titled NEMA MG 1 [20] and Part 10 in the document shows the minimum information that should be displayed on the nameplate for DC motors. The nameplate convention of motors is an industry practice in the US that allows motors that are encountered in the field to be quickly categorized. The MG 1 document is the standard that is used for many practical considerations for all types of motors that vary from very common issues to more obscure ones (e.g. Section 14.9 discusses how to prevent rodents from entering large motors). The MG 1 reference [20] that is provided is an older version of the standard that is available online. The latest version of the MG 1 standard was published in 2016. Details of the changes and ordering information are shown at the following references [21, 22].

(a) (b) (c)

Figure 6.5: (a) Example of an industrial Permanent Magnet DC motor, (b) Motor brush, (c) Motor nameplate. The nameplate of the motor in Figure 6.5c shows that it is a Leeson # CM31D17NZ2GC. This motor is similar to this motor: https://www.grainger.com/product/LEESON-1-3-HP-DC-Permanent-Magnet-48ZG52

Both motors have a maximum horsepower of 1/3 hp. (**Note:** *1 hp = 745.7 Watts*). Table 6.1 shows a list of different types of motors, along with their typical hp range. Commutation is required in DC motors so that the induced force in Equation 6.3 will not cancel itself out each time the motor rotates 180^o. The device that implements commutation is called the commutator [3]. A two segment commutator is a split ring device that causes the polarity of the current to change each time it crosses the split, which results in θ in Equation 6.1 to never go above 180^o. This polarity reversal causes the force (and torque) to continually produce a rotation in the same direction (shown in Figure 6.6) instead of alternating back and forth each half period.



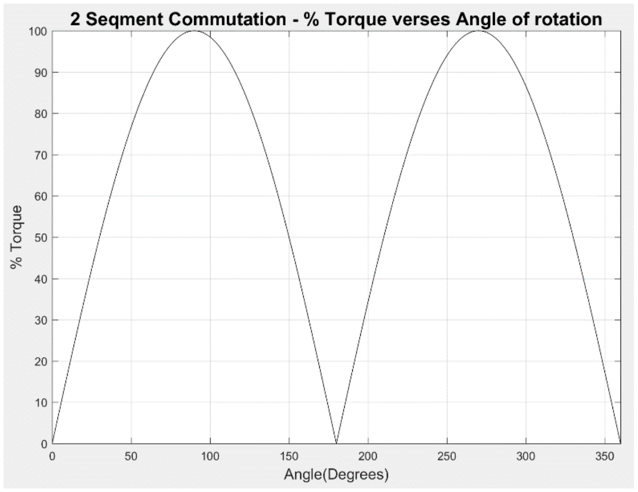


Figure 6.6: (Left) DC motor with 2-segment Commutator (Orange), Armature coil wires (purple), and 2-pole PM stator (N=Purple, S=Red). © Abnormaal. Used under CC BY-SA license, https://en.wikipedia.org/wiki/Brushed_DC_electric_motor#/media/File:Electric_motor.gif **(Right) The Torque vs angle of a DC motor with commutation.**

With no commutation (like the illustrations in Figures 6.3 and 6.4) the %Torque versus angle would look like a normal sinusoid that has a negative cycle between 180^o and 270^o. Torque is proportional to force and is at its maximum value (100% in Figure 6.6) when the armature wire and the field are perpendicular (i.e. at angles of 90^o and 270^o). While adding commutation is a big improvement over the motor in Figures 6.2 and 6.3 that couldn't even continually rotate, adding more commutation segments results in higher average torque. For example, Figure 6.7 shows the rotor of a small PM-DC motor that has 6 commutation segments and uses metal contacts as the brushes. The %Torque curve for this motor would result in a much more constant torque.

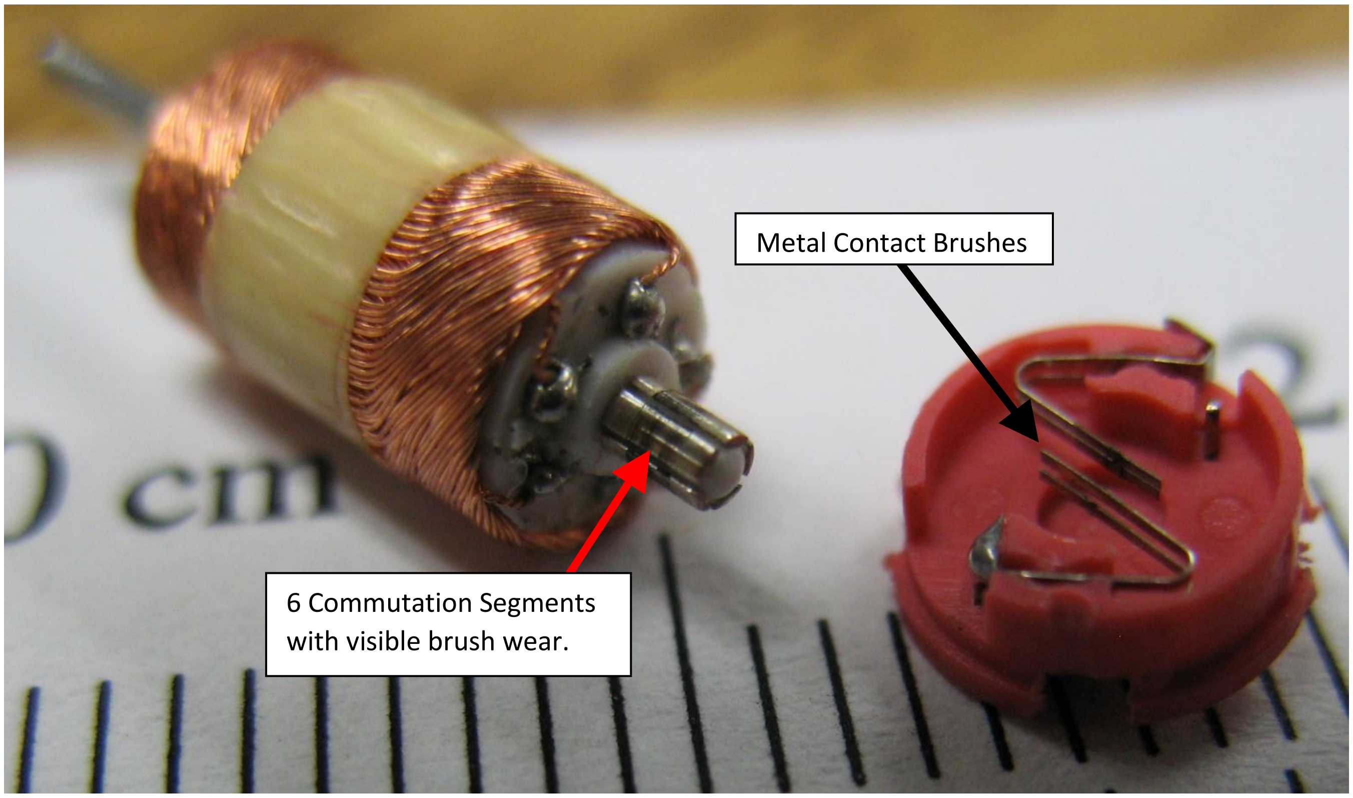


Figure 6.7: Commutator with 6 commutation segments in a small RC toy car PM-DC Motor. © Dale Mahalko. Used under CC BY-SA license: **https://en.wikipedia.org/wiki/Electric_motor#/media/File:Tiny_motor_windings_-_commutator_-_brushes_in_Zip_Zaps_toy_R-C_car.jpg**

The % Torque versus angle plots are shown on page 3-2 of the Bodine Gear Motor Handbook for 4-segment and 32-segment commutators [1]. This handbook is a great source to learn more about motors and will be used as a reference many times in this module. Figure 6.8 shows an 8 Pole "Salient pole" rotor and how each of the 8 commutation segment are wound. There is also a non-salient pole style of rotor that doesn't have its windings wrapped below the face of the pole [4].

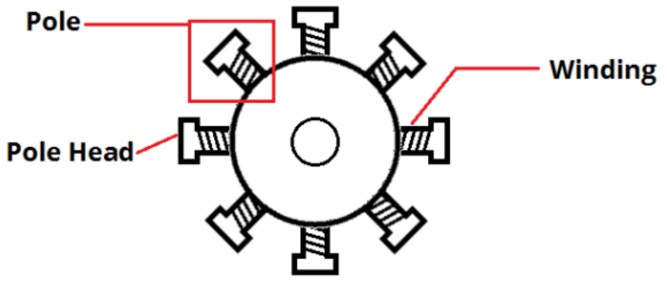


Figure 6.8: 8-Pole Salient pole rotor. © Sumwon1234. Used under CC BY-SA licence: https://en.wikipedia.org/wiki/Electric_motor#/media/File:Salient-pole_rotor.png

Table 6.1 shows an overview of many different types of motors. Additional details of three specific classes of motors are discussed in Sections 6.2 (PM-DC

Motors), 6.3 (Wound Field DC Motors), and 6.4 (AC Motors).

Before moving on to specific types of motors, equations that are applicable to almost all motors are shown.

[6.4] $\boldsymbol{Power\ (Mech.) = P_{out} = T \cdot \omega}$ * If T is in **N-m** and ω is in **rad/s**, then the output power is in **Watts**.

- T is torque in Newton-meters (N-m)
- ω is angular velocity in radians per second (rad/sec)

Note: *An important measure of mechanical output power is horsepower (hp).* **1 hp = 745.7 Watts**

When trying to determine the mechanical output power of a motor, the angular velocity (rad/sec) is used. However, rpm is a more common term that is used to indicate the speed of a motor. The conversion is:

[6.5] $\boldsymbol{\omega = \frac{2\pi}{60} \cdot n}$ * n is the speed in rpm and ω is the angular velocity in rad/sec.

The input power of a motor in Equation 6.6 is simply the basic electrical power formula.

[6.6] $\boldsymbol{Power\ (Elec.) = P_{in} = V \cdot I}$ * If V is in **Volts** and I is in **Amps** then the input power is also in **Watts**.

If the motor is a DC motor then V and I are constant DC values, but if it is an AC motor then RMS values are used. If a motor has both stator and armature windings, then the total input power must include both. For PM-DC motors the input power is simply the DC voltage applied multiplied by the armature DC current.

P_{out} must be less than P_{in} due to energy loss in the motor. The efficiency, η, of **any motor** can be calculated as:

[6.7] $\boldsymbol{\eta = P_{out}/P_{in}}$ * This results in a fraction, but it is usually multiplied by 100 to give a percentage.

One of the primary differences in the different types of motors is their speed vs torque curves. If the speed doesn't change very much as the torque increases the speed vs torque curve is flatter and the motor is said to have a good speed regulation. Equation 6.8 defines the relationship for speed regulation, as shown in equation 3.9 in reference 30. Both Equations 6.8 and 6.9, but can be multiplied by 100 to convert to a percentage.

[6.8] $\boldsymbol{Speed\ Regulation = SR = (n_0 - n_R)/n_R}$ Where: n_0 = No load speed & n_R = Speed at rated load.

Generators also have regulation issues. Ideally, a generator would produce the same voltage regardless of the load and would have a 0% voltage regulation, according to equation 3.10 in reference 30 (Equation 6.9 below).

[6.9] $\boldsymbol{Voltage\ Regulation = VR = (V_0 - V_R)/V_R}$ Where: V_0 = No load voltage & V_R = voltage at rated load.

Example 6.2) *A motor's rotor spins at 1,000 rpm when no load is applied (i.e. torque = 0), but reduces to 932 rpm when the rated torque of 2.37 N-m is applied. Determine the speed regulation and output horsepower at the rated load condition. If the motor's input power is 500 Watts at the rated load condition, what is the efficiency?* ***Note:*** *The efficiency should be at its maximum at the rated load condition.*

Solution)

From [6.5] $\omega = \frac{2\pi}{60} \cdot n = \frac{2\pi}{60} \cdot 932$ = **97.6 rad/sec**

From [6.4] Pout = T·ω = 2.37 N-m * 97.6 rad/sec = 231.3 W → 231.3 W * (1 hp/745.7 Watts) = **0.31 hp**

From [6.7] $\eta = P_{out}/P_{in}$ = 231.3/500 = **46.3%**

From [6.8] $SR = (n_0 - n_R)/n_R$ = (1000 – 932)/932 = **7.3%**

Table 6.1 provides an overview of some of the most common types of motors. More details of some of these are shown in Section 6.2 – PM-DC Motors, Section 6.3 – Wound Field DC Motors, and Section 6.4 – AC Motors.

Table 6.1) Overview of Motor Types. Unless otherwise cited, information in this Table is from reference [1].

Popular Motors	hp [5]	Summary and Notes
Section 6.2 Permanent Magnet DC (PM-DC)	1/26 - 10	The brushed DC motors (wound or PM field) in sections 6.2 and 6.3 require commutation to produce a net torque greater than zero. **Pros:** They are smaller, lighter, cheaper and much simpler than wound field motors. They are also more efficient because they don't have stator windings that have I^2R losses. Since they have a near linear speed vs torque curve design calculations are much easier to do. **Cons:** The stator magnet can become demagnetized. PMs produce a much weaker field than a wound stator (i.e. lower power).
Section 6.3 Series Wound DC (Universal)	10 - 200	They are often called self-excited because the current going through the stator field winding is the same current that goes through the armature, as shown in Figure 6.11B. **Pros:** They are also called Universal Motors because they can operate on DC or AC supplies. They can operate at high rpm and have the highest horsepower per pound of any motor that can operate from single phase AC power [1]. **Cons:** In Series wound motors, the speed vs torque curve drops rapidly as torque increases so they are not good at speed regulation. They operate at high rpm, which results in more bearing and brush damage.
Section 6.3 Shunt Wound DC	10 - 200	Since the field and armature windings are in parallel (i.e. shunt), the currents going through them are different, as shown in Figure 6.11A. They operate in the same way as separately-excited DC motors that have separate supply voltages for the field and the armature. **Pros:** They have a nearly flat speed vs torque curve so they have the best speed regulation of any motor and are easily reversed since the stator field can be controlled separately from the armature. **Cons:** They have lower rpm and lower starting torque than series wound motors.
Section 6.3 Compound Wound DC	10 - 200	The stator field windings are wired in both shunt and series with the armature windings, as shown in Figure 6.11C. They have a speed vs torque curve that is in between the shunt and series wound DC motors and is closer to the linear speed vs torque in a PM-DC motor [31].
Section 6.4.1 Three Phase AC Synchronous	1,000 - 50,000	This motor runs on 3-phase power and has a constant speed rotor that stays in sync with the rotating magnetic stator field. It has the highest hp due to the increased efficiency that comes with 3-phase power and synchronous motors. Precautions are needed to start it.
Section 6.4.1 Three Phase AC Induction	1 - 5,000	This motor operates like a synchronous motor that has a rotating magnetic field in the stator, but its rotor speed doesn't stay in sync with stator field due to slip. They don't require a starting mechanism like single phase motors do, but are much more expensive.
Section 6.4.2 Single Phase AC Induction	0.33 - 5	Runs on single phase AC power. **Pros** (Compared to Universal motors): Higher rated torque, much better speed regulation, quieter and less maintenance required because no brushes are used. **Cons:** Lower rpm. Very low starting torque so it requires a starting mechanism.
Brushless DC	5 - 14 [6]	Uses complicated circuitry to perform the function of commutation externally. **Pros:** Since they have no brushes (and no arcing) they are good to use if you are in an explosive environment and can't have arcing. They are also good when small contaminates might affect brushes. They also last much longer than brushed motors so they are good to use when replacing a motor is difficult. **Cons:** Much more expensive than brushed DC motors.
Servo Example 3.10 shows how to control a servo with LabVIEW using a Counter output.	Varies [7]	A servo motor contains a motor and an internal encoder with control circuitry. Inexpensive servos like the NXT motor in Figure 4.25 or this Parallax type [15] are made with a **PM-DC motor** and internal plastic gearing to increase torque and reduce speed [see Equation 6.16]. There are two types: Position servos rotate to a set position as different pulse widths are sent to the motor. They usually have either 90^o or 180^o ranges. Continuous servos rotate at different speeds as different pulse widths are sent to the motor. The Parallax servo [15] requires a square wave pulse with a period of ~ 20 ms and a high time that can be adjusted between 1.3 and 1.7 ms to change the speed and direction. 1.3 ms = full speed CW, 1.5 ms = stop, and 1.7 ms = full speed CCW. The 180^o version works in the same way, but the position is adjusted by changing the high time in the range of 0.75 to 2.25 ms [16].
Stepper	Varies [7]	Digital pulses are sent from circuitry or a microcontroller to electromagnets that are arranged inside the motor to produce precise incremental motion. **Pros:** Excellent position accuracy, good holding torque, Highly reliable. **Cons:** Speed is limited to step distance, high torque values can result in steps being skipped, maximum current is always drawn. [8, 10]

Section 6.2 – Permanent Magnet (PM) DC Motors

Since permanent magnet (PM) DC motors are far more commonly used for smaller applications that are often done by college students (e.g. the Robot on the cover page), this book will emphasize them more than the other types shown in Table 6.1. Photos of PM-DC motors are shown in Figures 6.5 and 6.7. PM-DC motors are different than most motors because the external magnetic field in the stator is caused by permanent magnets instead of electromagnets. Figure 6.9 shows a permanent magnet stator side by side with a wound stator. The wound stator has 10 **salient** (or protruding) poles. While the permanent magnet stator is much smaller, lighter and more efficient (because there are no windings and subsequently no I^2R losses through the windings), the permanent magnet produces a much weaker magnetic field and therefore results in lower output power for PM-DC motors. Since the output power is reduced, the combination of speed and torque is also reduced as shown in Equation 6.4. Another issue with PM-DC motors is that magnets in the stator can become demagnetized.

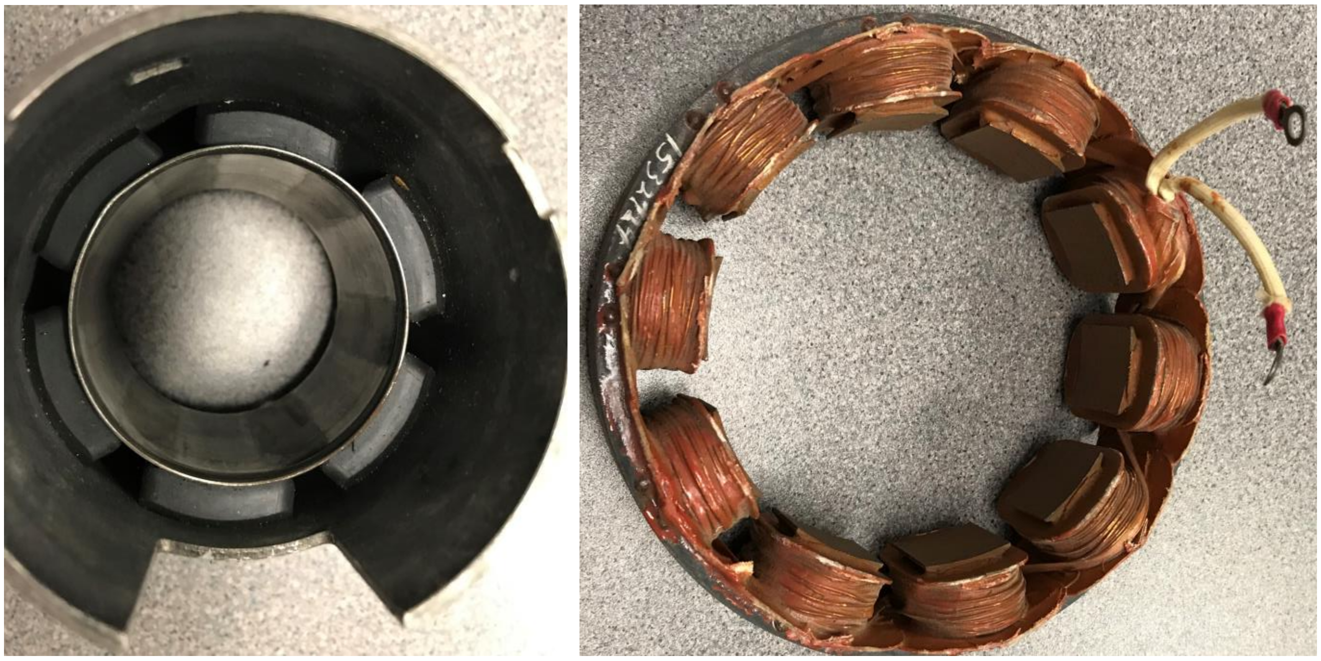


Figure 6.9) (a) 6-Pole stator with permanent magnets, (b) 10-pole wound field stator.

Efficiency is more difficult to calculate with wound field motors due to the external field also requiring power. Not only is efficiency much easier to calculate for PM-DC motors, but the **speed versus torque curve** is also much easier to produce because it can approximated as a linear equation of the form y = mx + b [36].

- The **x-axis** is torque. For small motors, gram-centimeters (g-cm) is often used as the unit of measure, but if T is converted to N-m, the torque is in the correct units to get the units of Watts for output power.
- The **y-axis** of the equation is usually given in rotational speed (rpm). If it is converted to angular velocity (rad/sec) then the output power can be easily determined from Equation 6.4. For this reason, rpm will be converted to rad/sec using Equation 6.5 anytime power and efficiency are calculated. Equation 6.10 is listed in terms of speed versus torque (n vs T) and in angular velocity versus torque (ω vs T).
- The **y-intercept (b)** is determined when there is "no load" (i.e. no torque) on the motor. This is often referred to as the **no load speed** in rpm (n_0) or **no-load angular velocity** in rad/sec (ω_0).

- The **slope (m)** of a linear equation is determined by the rise over run. In this case, the rise is the no-load speed and the run is the Stall Torque (T_S), which is the amount of torque where the motor stalls (i.e. stops rotating due to too much torque). If speed is converted to angular velocity, the slope is equal to: $\mathbf{m = -n_0/T_S}$. **Note:** *T_S is also referred to as the "locked rotor" torque since the rotor stops moving.*

The linear equation of the speed (or angular velocity) versus torque is:

[6.10] $n(T) = \frac{-n_0}{T_S} \cdot T + n_0$ **or if rpm is converted to rad/sec →** $\omega(T) = \frac{-\omega_0}{T_S} \cdot T + \omega_0$

- n(T) is the actual rotational speed in rpm. ω(T) is the actual angular velocity in rad/sec.
- $\mathbf{n_0}$ is the no load speed. $\boldsymbol{\omega_0}$ is the no-load angular velocity. This is where no load is on the shaft (i.e. T = 0).
- T is the applied (or actual) torque on the motor shaft in Newton-meters (N-m).
- T_S is the amount of torque required to "stall" or stop the motor from rotating (i.e. n = ω = 0).

The equation for the current through the armature versus applied torque in a PM-DC motor can also be approximated as a linear equation, but in this case the slope will be positive because **torque increases as current increases**. When no torque is applied and the motor is in the no-load speed condition, very little current flows through the motor, but when the motor is stalled (i.e. locked rotor condition) the current is at its maximum. It is important to never operate a motor near its stall torque for an extended period of time because the excessive current will damage the motor.

Note: *Most PM-DC motors are designed to operate at torques less than 25% of the Stall Torque ($0.25 \cdot T_S$). The torque at which the motor is designed for is called the "**rated load**" and it is the torque at which the motor is expected to produce its maximum efficiency and therefore the ideal operating point. The rated load is usually ranges from $0.15 \cdot T_S$ and $0.25 \cdot T_S$, depending on the motor. The efficiency at the rated load is an important consideration when selecting a motor.*

[6.11] $I(T) = \frac{(I_S - I_0)}{T_S} \cdot T + I_0$

- I(T) is the current (in Amps) that is supplied to the armature of a PM-DC motor.
- I_0 is the current when no load is on the shaft (i.e. T = 0). It is often referred to as No-Load Current.
- I_S is the current when the motor stalls (i.e. n = ω = 0). It is often referred to as Stall Current.
- T is the applied (or actual) torque on the motor shaft in Newton-meters (N-m)
- T_S is the amount of torque required to "stall" or stop the motor from rotating (i.e. n = ω = 0).

Using equations 6.4 to 6.7 and 6.10 to 6.11, plots can be created for current, speed (or angular velocity), output power, and efficiency as the torque varies from 0 to T_S. These four plots will be referred to as the "Motor Curves" and are shown in Figure 6.10. The datasheet for PM-DC motors will sometimes have these curves included. Keep in mind for PM-DC motors, these plots come from linear approximations. To more accurately model a PM-DC motor, it involves a complicated procedure that involves factoring internal motor inductance into the equation. An example of an attempt to more accurately model a PM-DC motor is described in reference [19]. While it depends on the situation, the Motor Curves that come from linear approximations are usually sufficient when selecting/sizing a PM-DC motor for an application so the linear model is used in this book. The following example shows how to use equations 6.4 to 6.7 and 6.10 to 6.11 to produce the motor curves. The values in Figure 6.10 were specifically selected so that all of the curves can be plotted on the same graph without having to use multiple y-axes. Typically, putting all of these quantities on a common axis won't produce results that are easy to analyze, so separating them on different y-axes scales or doing different plots for ω vs T, P_{out} vs T, I vs. T, and η vs T can be done (e.g. see plots in Example 6.5).

*	Nominal Voltage	16	V
1)	No Load ω (ω_0)	20	rad/sec
2)	Stall Torque (T_s)	10	N-m
3)	No Load Current (I_0)	1	A
4)	Stall Current (I_s)	21	A

* The voltage is Fixed. If the voltage changes, the output power and efficiency values change.
Determine control variables 1) through 4) experimentally or from the motor datasheet.
The values in this example are not realistic. They were selected so only one scale would be needed for the y-axis.
- To simplify this example the y-axis is in rad/sec instead of having the more commonly used speed with RPM units on the y-axis. Also, N-m is used for Torque on the plots instead of gram-cm.

(ω vs. T) equation: $\omega(T) = [-\omega_0/Ts]*T + \omega_0$ (This is a linear equation: y = mx+b)
(I vs. T) equation: $I(T) = [(I_S - I_0)/(Ts)]*T + I_0$ (max Current is at the stall torque)

	TABLE - Created From Eqs. [6.4] to [6.11]			$P_{in} = V*I$	$P_{out} = T*\omega$	$\eta = P_{out}/P_{in}$
	Torque (N-m)	ω (rad/sec)	I (Amps)	P_{in} (Watts)	P_{out}(Watts)	Efficiency %
ω_0	0	20	1	16	0.0	0
	0.5	19	2	32	9.5	30
	1	18	3	48	18.0	38
	1.5	17	4	64	25.5	40
η_{max}	2	16	5	80	32.0	**40**
	2.5	15	6	96	37.5	39
	3	14	7	112	42.0	38
	3.5	13	8	128	45.5	36
	4	12	9	144	48.0	33
	4.5	11	10	160	49.5	31
P_{max}	5	10	11	176	**50.0**	28
	5.5	9	12	192	49.5	26
	6	8	13	208	48.0	23
	6.5	7	14	224	45.5	20
	7	6	15	240	42.0	18
	7.5	5	16	256	37.5	15
	8	4	17	272	32.0	12
	8.5	3	18	288	25.5	9
	9	2	19	304	18.0	6
	9.5	1	20	320	9.5	3
T_S	10	0	21	336	0.0	0

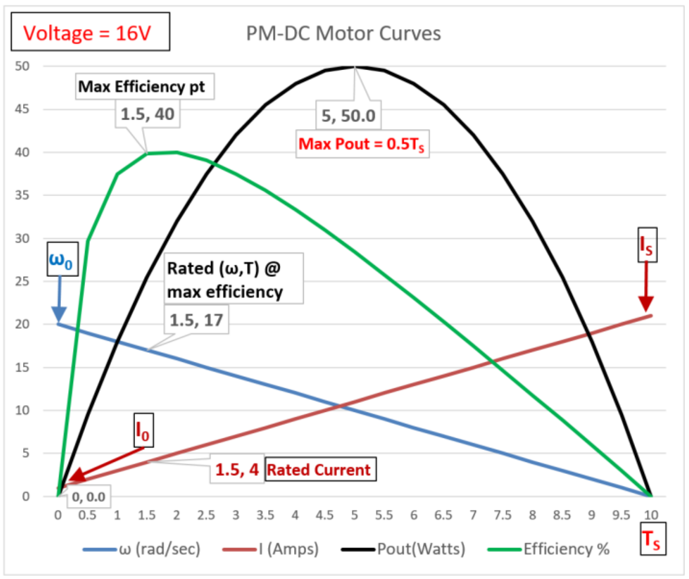


Figure 6.10: PM-DC Motor Curves and Calculations

To create the table of data in Figure 6.10, the number of data points for the curve needs to first be selected and this number will set the resolution of the torque. In Figure 6.10, a torque resolution of 0.5 N-m per step was used which resulted in 21 steps when plotting from 0 to T_S. If the results need to be known at a smaller torque resolution, then more data points need to be included. The data in Figure 6.10 includes hypothetical values to demonstrate how the process works, but Example 6.5 will create the Motor Curves for an actual motor [17].

There are two other important equations when dealing with motors that are shown in Equation 6.12 and 6.13 [23]. Equation 6.12 allows the angular velocity to be converted to linear velocity. For a motorized vehicle, the radius of the wheel is equal to **r** and it can be used to set the velocity of the vehicle if ω is known. In Equation 6.13, the radius of the wheel is used to convert force to torque. Once the forces are summed on the motor, it can be multiplied by the wheel radius to determine the actual torque that is applied by the motor and this torque value can be used to determine the speed, power, current, and efficiency.

[6.12] $v = \omega \cdot r$ * Where **v** = linear velocity in m/s and **r** = radius of wheel in meters.

[6.13] $T = F \cdot r$ * Where **F** = Force on motor in N and **r** = radius of wheel in meters.

The process of creating the Motor Curves and solving problems with them (such as selecting a motor for an application) is done by the following steps. In step 3, PM-DC motor problems are broken into types A, B, and C.

1) **Determine ω_0 (or n_0), T_S, I_0, I_S, and the nominal voltage** from the motor datasheet or experimentally.
2) **Customize Equation 6.10** with the values obtained for ω_0 (or n_0) and T_S.
3) **Determine the mechanical torque required** in the application and solve for ω. Some options for this are:

A. The applied (or actual) torque is given in the problem statement. Use that value of torque and **Equation 6.10** to determine the values of ω. This “type-A” problem is the only type that is covered in this book.

B. The required power can be determined from the equation **Power = Work/Time = Force*Distance/Time** and then the value of required power is used to get torque in terms of ω. Once power is determined, use **Equation 6.10** and **T = P/ω** to determine the values of T and ω. An example of a problem like this would be a PM-DC motor that is used with a frictionless pulley to lift a weight in a set amount of time.

C. For robot vehicles, torque can be computed from Equation 6.13, where the distance is equal to the radius of the wheel and the forces can be determined from doing a free body diagram of the vehicle. Reference [23] provides some background that will help determine the forces on a vehicle.

4) **Use I_0, I_S to customize Equation 6.11**. Use the torque value from 3) to determine the actual motor current.

5) Use I, T, ω, and voltage to **determine** input power, output power, and the motor **efficiency**.

6) Once the first 5 steps are complete, the speed vs torque curve can be shifted left or right based on the voltage. As the voltage increases the slope will remain approximately the same, but the speed and torque will increase (i.e. ω vs T curve shifts to the right). The no-load speed at the new voltage needs to be known (either experimentally or from the datasheet) to get the new ω vs T relationship as shown in Equation 6.14.

[6.14] $\boldsymbol{\omega(T) = \frac{-\omega_{01}}{T_{S1}} \cdot T + \omega_{02}}$ * This is the ω vs T relationship when the voltage changes from nominal.

- ω_{01} is the angular velocity when no load is on the shaft at the nominal voltage.
- T_{S1} is the amount of torque required to “stall” or stop the motor from rotating at the nominal voltage.
- ω(T) is the actual angular velocity of the motor shaft at the new voltage.
- T is the applied (or actual) torque on the motor shaft at the new voltage.
- ω_{02} is the angular velocity when no load is on the shaft at a new voltage.

Example 6.3) *This is a type-A Problem with an applied torque of 0.004 N-m. If the values below are obtained from the data sheet, calculate the angular velocity, input & output power, and efficiency of the PM-DC motor.*

- *Nominal Voltage = 18 V*
- *No Load Speed = 20,000 rpm (**Note:** This will need to be converted to rad/sec to solve for ω)*
- *Stall Torque = 0.02 N-m*
- *No Load Current = 100 mA*
- *Stall Current = 3.1 A*

Solution) First, the units need to be put in rad/s and N-m so Watts are the output power units.

From Equation 6.5 → ω_0 = 20000 rpm * (2π/60) = **2094.395 rad/sec**

From [6.10] $\omega(T) = \frac{-\omega_0}{T_S} \cdot T + \omega_0 = \frac{-2094.395}{0.02} \cdot (0.004) + 2094.395 = \mathbf{1675.5\ rad/sec}$

From [6.11] $I(T) = \frac{(I_S - I_0)}{T_S} \cdot T + I_0 = \frac{3.1-0.1}{0.02} \cdot T + 0.1 = \frac{3}{0.02} \cdot 0.004 + 0.1 = \mathbf{0.7\ A}$

From [6.4] $P_{out} = T \cdot \omega = 0.004 \cdot 1675.5$ = **6.7 Watts**

From [6.6] $P_{in} = V \cdot I = 18 \cdot 0.7$ = **12.6 Watts**

From [6.7] $\eta = P_{out}/P_{in} = 6.7/12.6$ = **0.532 or 53.2%**

Example 6.4) *If the voltage of the motor in Example 6.3 was decreased to 15V and the new no load speed was measured with a tachometer to be 16,500 rpm determine the angular velocity when the torque is 0.004 N-m and plot ω versus T and compare it to ω versus T when the voltage is 18V. If two of these motors were used to power a vehicle that had wheel diameters of 20 cm, what is the net force and linear velocity of the vehicle at each of these two ω vs T operating conditions?*

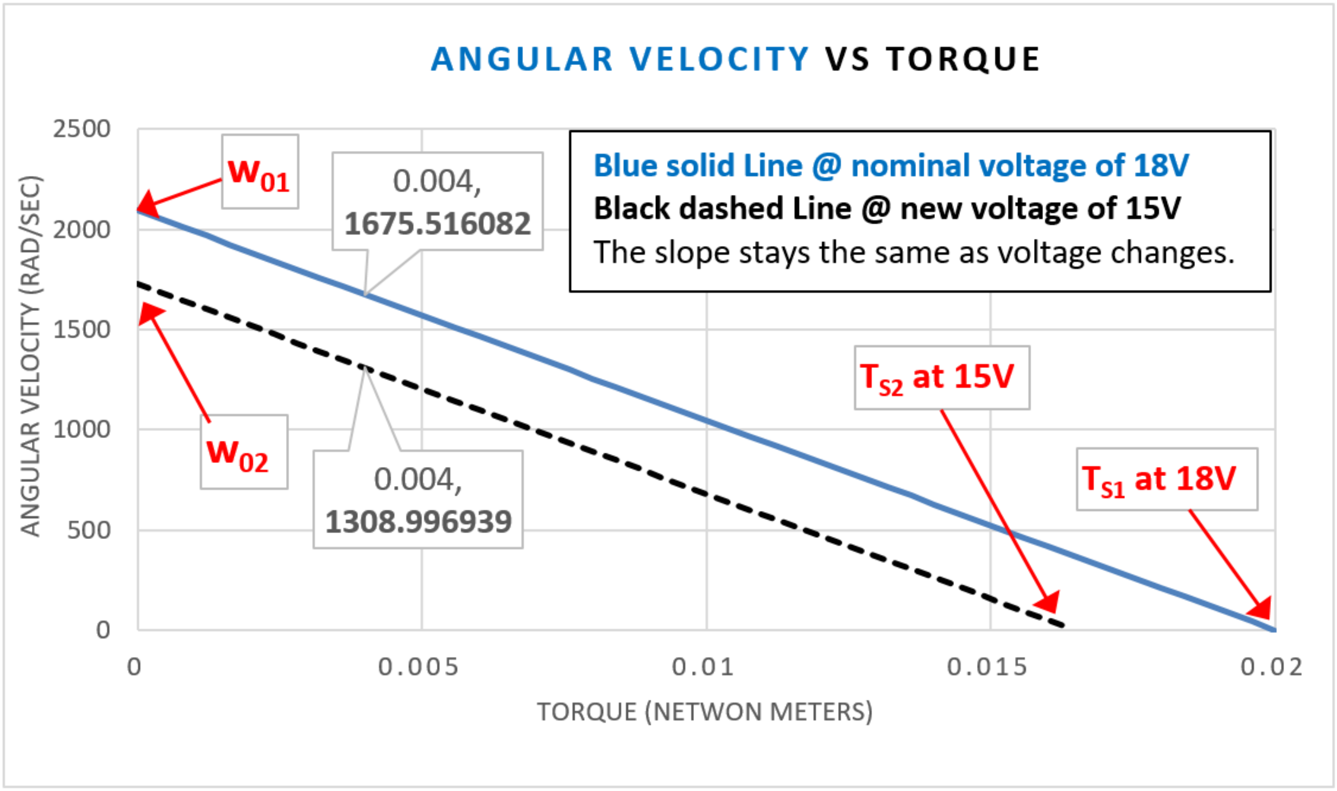
Solution) First the units for ω_0 needs to be put in rad/s.
From Equation 6.5 → ω_{02} = 16,500 rpm * (2π/60) = **1727.9 rad/sec**

[6.14] $\omega(T) = \frac{-\omega_{01}}{T_{S1}} \cdot T + \omega_{02} = \frac{-2094.35}{0.02} \cdot 0.004 + 1727.9$ = **1309 rad/sec** *ω_{01} and T_{S1} given in Example 6.3

This answer is verified in the Figure below.
Since both of these motors have the same angular velocity, the vehicle should move in a straight path at the following velocity, v, as long as the caster/support device is symmetrical and not preventing motion in any direction and the wheels are identical and the road surface is the same for both wheels.

- @ 15 V: $v = \omega \cdot r$ = 1309 rad/s · (0.02m/2) = **13.09 m/s** (This is equal to 29.3 miles per hour)
- @ 18 V: $v = \omega \cdot r$ = 1675.5 rad/s · (0.02m/2) = **16.755 m/s** (This is equal to 37.5 miles per hour)

Since the torque is constant in this problem for both voltage levels, the net force on each of the 2 motors is:
$T = F \cdot r$ → F = T/r = 0.004/(0.02m/2) = **0.4 N** (The total force on the entire vehicle would be **0.8 N** (2 · 0.4 N)



Example 6.5) *A RF-370CA Mabuchi PM-DC Motor from Jameco provides the following parameters in the datasheet [17].* Verify that the rated values from the datasheet (* **shown below**) match the calculations.

No-Load Speed (n_0) = 5,600 rpm	Stall Torque (T_S) = 187 g-cm	Nominal Voltage = 12 V
No-Load Current (I_0) = 0.026A	Stall Current (I_S) = 1.06 A	

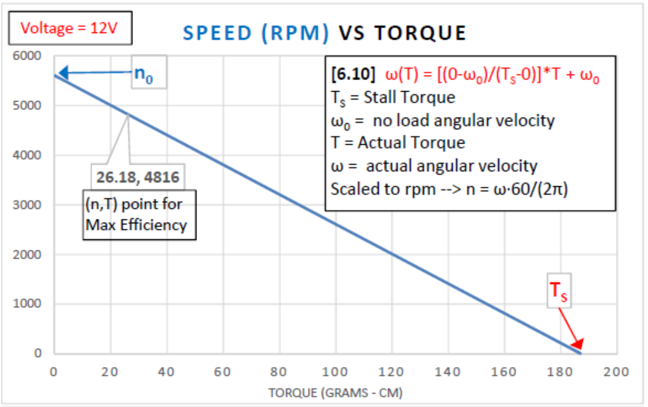
***Rated** Torque = <u>25.3g-cm</u> (The rated torque is the torque at η_{max}, and in the table below it is at **<u>26.18 g-cm</u>**)

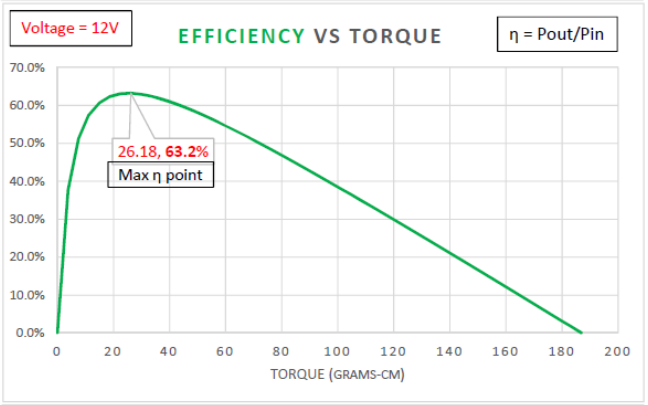
***Rated** Current = <u>0.17 A</u> → **I (@26.18 g-cm) = <u>0.1708 A</u>** ***Rated** P_{out} = <u>1.25 A</u> → **P_{out} (@26.18 g-cm) = <u>1.295 W</u>**

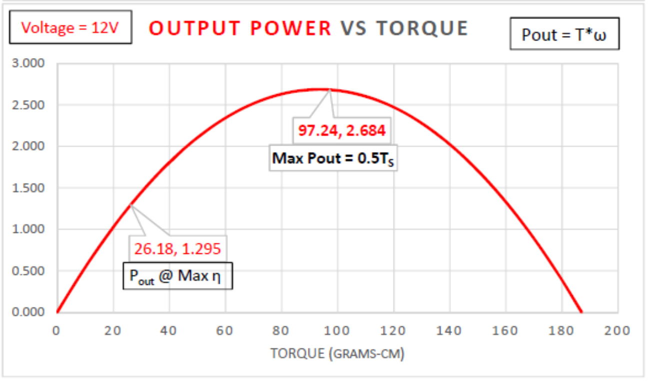
Solutions) The calculations and datasheet values come out very close. They are listed side by side above.

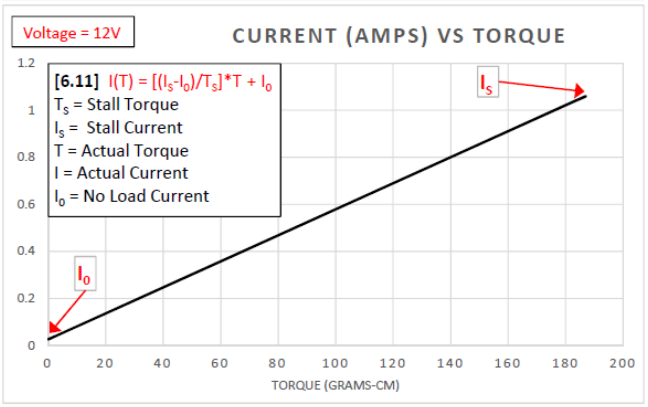
Nominal Voltage	12	V	<-- Value obtained from datasheet
No Load rpm (n_0)	**5600**	**rpm**	<-- Value obtained from datasheet
No Load ω (ω_0)	586.43	rad/sec	**rpm converted to rad/sec**
Stall Torque (T_S)	**187**	**g-cm**	<-- Value obtained from datasheet
Stall Torque (T_S)	0.0183	N-m	**g-cm converted to N-m**
No Load Current (I_0)	0.026	A	<-- Value obtained from datasheet
Stall Current (I_S)	1.06	A	<-- Value obtained from datasheet

	TABLE (Created From Equations 6.4 to 6.11)					**P_{in} = V*I**	**P_{out} = T*ω**	**η = P_{out}/P_{in}**
	T (g-cm)	T (N-m)	ω (rad/sec)	n (rpm)	I (Amps)	P_{in} (Watts)	P_{out}(Watts)	Efficiency
n_0	0.00	0	586.43	5600	0.026	0.312	0.000	0.0%
	3.74	0.00037	574.70	5488	0.0467	0.560	0.211	37.6%
	7.48	0.00073	562.97	5376	0.0674	0.808	0.413	51.1%
	11.22	0.00110	551.24	5264	0.088	1.056	0.607	57.4%
	14.96	0.00147	539.52	5152	0.1087	1.305	0.792	60.7%
	18.70	0.00183	527.79	5040	0.1294	1.553	0.968	62.3%
	22.44	0.00220	516.06	4928	0.1501	1.801	1.136	63.1%
η_{max}	26.18	0.00257	504.33	4816	0.1708	2.049	1.295	**63.2%**
	29.92	0.00293	492.60	4704	0.1914	2.297	1.445	62.9%
	33.66	0.00330	480.87	4592	0.2121	2.545	1.587	62.4%
	37.40	0.00367	469.14	4480	0.2328	2.794	1.721	61.6%
	41.14	0.00403	457.42	4368	0.2535	3.042	1.845	60.7%
	44.88	0.00440	445.69	4256	0.2742	3.290	1.962	59.6%
	48.62	0.00477	433.96	4144	0.2948	3.538	2.069	58.5%
	52.36	0.00513	422.23	4032	0.3155	3.786	2.168	57.3%
	56.10	0.00550	410.50	3920	0.3362	4.034	2.258	56.0%
	59.84	0.00587	398.77	3808	0.3569	4.283	2.340	54.6%
	63.58	0.00624	387.04	3696	0.3776	4.531	2.413	53.3%
	67.32	0.00660	375.32	3584	0.3982	4.779	2.478	51.8%
	71.06	0.00697	363.59	3472	0.4189	5.027	2.534	50.4%
	74.80	0.00734	351.86	3360	0.4396	5.275	2.581	48.9%
	78.54	0.00770	340.13	3248	0.4603	5.523	2.620	47.4%
	82.28	0.00807	328.40	3136	0.481	5.772	2.650	45.9%
	86.02	0.00844	316.67	3024	0.5016	6.020	2.671	44.4%
	89.76	0.00880	304.94	2912	0.5223	6.268	2.684	42.8%
P_{max}	93.50	0.00917	293.22	2800	0.543	6.516	**2.689**	41.3%
	97.24	0.00954	281.49	2688	0.5637	6.764	2.684	39.7%
	100.98	0.00990	269.76	2576	0.5844	7.012	2.671	38.1%
	104.72	0.01027	258.03	2464	0.605	7.260	2.650	36.5%
	108.46	0.01064	246.30	2352	0.6257	7.509	2.620	34.9%
	112.20	0.01100	234.57	2240	0.6464	7.757	2.581	33.3%
	115.94	0.01137	222.84	2128	0.6671	8.005	2.534	31.7%
	119.68	0.01174	211.12	2016	0.6878	8.253	2.478	30.0%
	123.42	0.01210	199.39	1904	0.7084	8.501	2.413	28.4%
	127.16	0.01247	187.66	1792	0.7291	8.749	2.340	26.7%
	130.90	0.01284	175.93	1680	0.7498	8.998	2.258	25.1%
	134.64	0.01320	164.20	1568	0.7705	9.246	2.168	23.4%
	138.38	0.01357	152.47	1456	0.7912	9.494	2.069	21.8%
	142.12	0.01394	140.74	1344	0.8118	9.742	1.962	20.1%
	145.86	0.01430	129.01	1232	0.8325	9.990	1.845	18.5%
	149.60	0.01467	117.29	1120	0.8532	10.238	1.721	16.8%
	153.34	0.01504	105.56	1008	0.8739	10.487	1.587	15.1%
	157.08	0.01540	93.83	896	0.8946	10.735	1.445	13.5%
	160.82	0.01577	82.10	784	0.9152	10.983	1.295	11.8%
	164.56	0.01614	70.37	672	0.9359	11.231	1.136	10.1%
	168.30	0.01650	58.64	560	0.9566	11.479	0.968	8.4%
	172.04	0.01687	46.91	448	0.9773	11.727	0.792	6.7%
	175.78	0.01724	35.19	336	0.998	11.976	0.607	5.1%
	179.52	0.01760	23.46	224	1.0186	12.224	0.413	3.4%
	183.26	0.01797	11.73	112	1.0393	12.472	0.211	1.7%
T_S	187.00	0.01834	0.00	0	1.06	12.720	0.000	0.0%









Example 6.6) *A Jameco MD5-1885 states that it has a maximum efficiency of **60.7%** in the datasheet [18]. If the following values are listed at maximum efficiency (rated values), verify that 60.7% agrees with these parameters.*
1) Speed @ Maximum Efficiency = 9820 rpm → w_0 = (9820 rev/min)(2π radians/rev)(1min/60s) = 1028.35 rad/sec
2) Torque @ Maximum Efficiency = 53.5 g-cm → T = (53.5 g-cm)(0.00980665 N/1g)*(.01m/1cm) = 0.0052466 N-m
3) Current @ Maximum Efficiency = 0.74 Amps & nominal voltage is 12 V (The solution is on the next line.)
Pin = V*I = 12*0.74 = 8.88 W **Pout** = T*ω = .0052466*1028.35 = 5.39 **η** = **Pout/Pin** = 5.39/8.88 = **60.7%**

Section 6.2.1 – PM-DC Speed Reduction Techniques and Gearing

One of the biggest problems with PM-DC motors is that they spin too fast and don't have enough torque for most applications. As shown in Equation 6.14, one way the speed can be reduced is by decreasing the voltage. PM-DC motors typically have a wide operating voltage range that is usually specified in the datasheet. There are three popular ways to reduce the voltage of a motor to slow it down:

1) Select a battery with a lower voltage.
2) Add a resistor in series with the battery so there is a voltage drop.
3) Use Pulse Width Modulation (PWM) to provide fine adjustment to the average voltage that is applied to the motor. When a motor is controlled by a PWM signal a square wave is sent to the motor that has the duty cycle (i.e. high_time/period) adjusted so that the average voltage (V_{avg} = duty cycle * peak voltage) is changed. As long as the frequency of the square wave isn't too low, the motors speed versus torque relationship will respond to the average voltage of the square wave as if it is a DC value. PWM is frequently implemented with software, but it can easily be implemented in hardware with a 555 timer IC [25, 26]. Figure 8 of the National Semiconductor LM555 datasheet shows how to connect a 555 Timer for PWM [27].

While the three methods above are effective at reducing the speed, the problem with reducing the voltage is that it also reduces the torque, as shown in the speed versus torque curve in Example 6.4. Using gears is the solution to this problem. Gears can either be added to the motor shaft externally or inside the motor housing to both reduce speed and increase torque. When multiple external gears are used, it is called a gear box and when there are internal gears inside a gear motor it is a called a gear head. When using gears, the speed versus torque curve is greatly affected by the gears. The speed (and angular velocity) decrease by the gear ratio, which is a ratio of the number of teeth of the gear on the destination shaft, N_2, and the number of teeth on the gear of the source shaft, N_1. If multiple gears are connected with each other (either externally or inside the motor) it is called a gear train. All of the individual gear ratios are multiplied together in a gear train to get the total gear ratio. An example of a gear train is shown in the photo of the Lego NXT motor in Figure 4.25a.

[6.15] $\boldsymbol{Gear\ Ratio = GR = \frac{N_2}{N_1}}$ * If GR > 1 it is said to be geared up so torque increases and ω decreases.

[6.16] $\boldsymbol{\omega_2 = \frac{\omega_1}{GR} = \frac{\omega_2 \cdot N_1}{N_2}}$ * Due to power loss in the gears, torque doesn't increase as much as ω decreases.

There are many type of gears and many techniques used to increase the efficiency of them. More information is provided in chapter 6 of reference [1]. The output power from Equation 6.4 can be broken down for a motor with gears, as shown in Equation 6.17 below. The same techniques described above to plot the speed versus torque curve for a PM-DC motor without gears can be used for geared PM-DC motors, but much lower overall efficiencies will result due to power loss in the gears.

[6.17] $\boldsymbol{P_{out} = T \cdot \omega = P_{in} - P_{motor\_loss} - P_{gear\_loss}}$

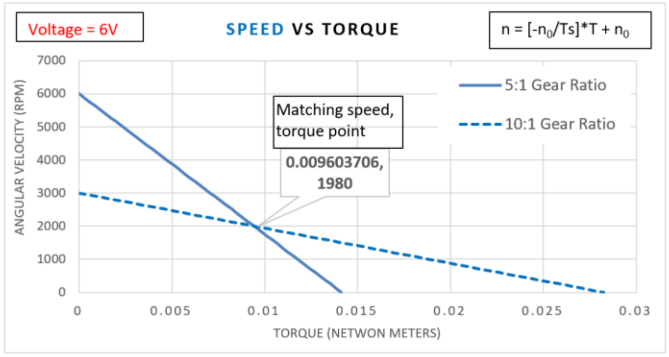
Caution: *Many small inexpensive gear motors use plastic gears (see Figure 4.25a). When using plastic gears it is critical that the motor not be operated beyond the rated condition for extended periods of time because the increased current will result in heat that can cause deformation or melting of the gears.*

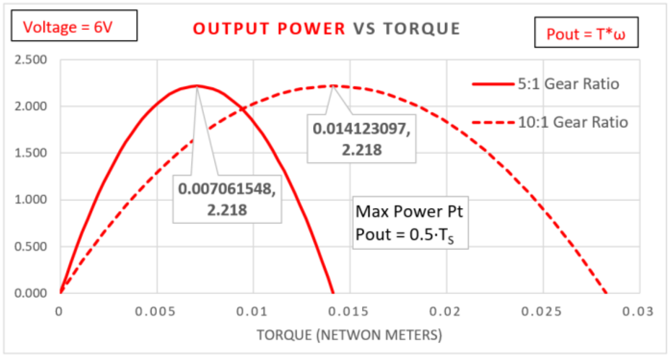
Example 6.7) *Select a motor that operates at 2,000 rpm and 0.1 N-m without **external** gears.*

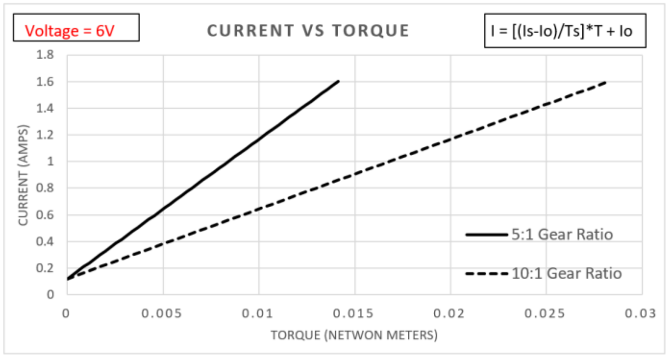
Solution) First, the Mabuchi motor in Example 6.5 will be tried. It has specifications: Ts = 0.01834 N-m & n_0 = 5600. From Equation 6.10, when torque is equal to 0.01 N-m, n is equal to **2,546 rpm**. Since this speed is greater than 2,000 rpm the speed would need to be reduced without decreasing the torque for this motor to be meet the specifications. The only way to make this motor work would be to use external gearing.
In order to achieve lower speeds without the use of external gears, a gear motor is likely needed. A small and inexpensive gear motor that has many different gear ratio options will be tried next. It is the Pololu Micro Metal Gear Motor model. The lowest two gear ratios will be tried since the speed required in this example is not that much lower then what can be achieved in a standard PM-DC motor like the Mabuchi. The datasheet shows:

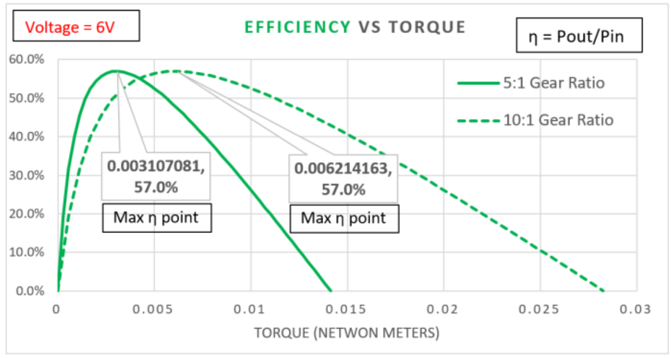
- 5:1 Gear Ratio Model: Voltage = 6 V, I_0 = 0.12 A, I_S = 1.6 A, n_0 = 6,000 rpm, T_S = 2 ounce-inches [28].
- 10:1 Gear Ratio Model: Voltage = 6 V, I_0 = 0.12 A, I_S = 1.6 A, n_0 = 3,000 rpm, T_S = 4 ounce-inches [29].

Note: *For this motor model, there are different options that have different voltage and current specifications.*
The motor curves are shown below for these two motor models.









The desired operating point was ~ 2000 rpm with a torque of ~ 0.01 N-m in this example. The speed-torque curve above shows that both motors n vs T linear equation is capable of operating at a point of ~ (2000 rpm, 0.01N-m). This condition just happens to be the exact point where the n vs T lines of the motors intersect in this example. When there are two similar motors that will work for an application, the one that is at the higher efficiency point should be selected. This point will also have lower current. In this example, the 5:1 gear ratio motor has an efficiency of ~ 27% and draws ~ 1.2 Amps at 0.01 N-m, but for the 10:1 gear ratio motor the efficiency is approximately doubled and the current is approximately cut in half. Therefore, **the 10:1 motor is the best motor to select of the choices presented** in this example.

Two gear ratios were selected for this example that were the two lowest available with this motor. This motor has 11 different gear ratio options and the highest is 1000:1. At the time of publication the price for the motor was $15.95 for each gear ratio option, except for the 1000:1 gear ratio, which was $22.95. The no load speed and stall torque of the 5:1 gear ratio is 6000 rpm and 2 oz.-in., respectively. When the gear ratio is increased to 1000:1, the no load speed and stall torque are 32 rpm and 125 ounce-inches, respectively. There is also a special note on the 1000:1 stall torque value that says that the 125 ounce-inch value is a theoretical value because stalling the gear motor would result in damaging the internal gears. A common practice is to avoid measuring the stall torque so that the motors aren't damaged due to high current and heat that is produced at the stall condition. If the speed and torque are measured at multiple torque settings, the linear equation of speed versus torque can be extrapolated to the point where the speed reaches zero (which is the stall torque condition).

Caution: *While the geared motors in this book are assumed to be linear, adding gears to a motor generally makes the linear speed versus torque approximation of a PM-DC motor less accurate than motors without gears.*

Section 6.2.2 – Characteristics of Lego Gear Motors

Figure 4.25 shows that the Lego NXT motor is another example of a geared motor. Reference [24] shows values for the no load speed & current and the stall torque & current that were experimentally determined for the Lego NXT, Lego EV3, and other types of Lego motors. An experiment to determine the loaded characteristics (i.e. speed, torque, and current) was set up using the Type B loading method that was previously described. It is assumed that the values obtained correspond to sending 100 (max power) to the motor in the power setting as shown in the LabVIEW VI in Figure 0.1. As the torque is held constant in the experiment, the data shows that the rpm increases nearly linearly as the voltage is increased from its 9 V nominal value. The Lego motor data also shows that while increasing the voltage the current stays nearly constant as long as the torque is kept constant. This is expected since current is proportional to torque in a motor. As the voltage was increased or decreased there was a nearly linear relationship between the speed and voltage for the Lego NXT, EV3, and many of the other motors. By obtaining the no load speed at another voltage, Equation 6.14 can be used along with the stall torque and no load speed at the nominal voltage to approximate the speed versus torque relationship at other voltages. The values in reference [24] are not expected to be at a high degree of accuracy and since these motors include internal gearing the speed versus torque curve will likely not perfectly linear. For example, the reference clearly states measuring the stall torque by manually locking the rotor is "VERY imprecise." However, even though the data in reference [24] is likely not extremely accurate, it shows a comparison of different motors and provides information about how to go about measuring motor characteristics.

Example 6.8) *Using the experimental data in reference 24, determine the linear and angular speed and the efficiency of an NXT motor when it is operating at 9 V, while a torque of 0.1 N-m is applied. Assume the wheels are used with the NXT motor that have a 2" diameter and there is no slipping or loss of traction so the angular velocity is converted to linear velocity from Equation 6.12. In addition to the Lego NXT motor, also solve this problem for the large Lego EV3 motor and solve them side by side so a direct comparison can be made. If the voltage is decreased to 8 V what is the new linear velocity?*

Solution) First, solve for the efficiency at **9 V** using the data in reference 24.

NXT: n_0 =170rpm, i_0 =60mA, T_S =0.5N-m, i_S = 2A	**EV3:** n_0 = 175rpm, i_0 = 60mA, T_S = 0.43N-m, i_S = 1.8A
n(**0.1** N-m) = - (170/0.5) **(0.1)**+170 = 136 rpm	n(**0.1** N-m) = - (175/0.43)(**0.1**)+175 = 134.3 rpm
ω(**0.1** N-m) = 136 rpm * 2π/60 = **14.24 rad/sec**	ω(**0.1** N-m) = 134.3 rpm * 2π/60 = **14.06 rad/sec**
Pout = T*ω = 0.1 N-m*14.24 rad/sec = 1.424 W	Pout = T*ω = 0.1 N-m*14.06 rad/sec = 1.406 W
i(**0.1** N-m) = ((2-0.06)/0.5)***(0.1)**+0.06 = 0.448A	i(**0.1** N-m) = ((2-0.06)/0.43)*(**0.1**)+0.06 = 0.511A
η = P_{out}/P_{in} = 1.424W/(9*0.448) = **35.3 %**	η = P_{out}/P_{in} = 1.406W/(9*0.511) = **30.6 %**

To find the linear velocity, convert the diameter in inches to radius in meters. d = 2” → r = 0.0254 m

v = ω · r = 14.24 rad/sec · 0.0254m = **0.362 m/s**

v = ω · r = 14.06 rad/sec · 0.0254m = **0.357 m/s**

Next, solve for the efficiency at **8 V** using the data in reference 24.
Assuming the slope of speed versus voltage is linear (which appears to be the case in the graphs) for the NXT and EV3 motors, the no load speed for the NXT is equal to 170*8/9 = 151.1 rpm and the no load speed for the EV3 is equal to 175*8/9 = 155.6 rpm. Using Equation 6.14 with the slope of the speed versus torque line determined previously by the conditions at 9 V, the following calculations can be made:

NXT:

n(**T_s**) = 0 = - (170/0.5) **(T_s)**+151.1 → T_s = 0.444 N-m

n(**0.1** N-m) = - (170/0.5) **(0.1)**+151.1 = 117.1 rpm

ω(**0.1** N-m) = 117.1 rpm * 2π/60 = **12.26 rad/sec**

v = ω · r = 12.26 rad/sec · 0.0254m = **0.311 m/s**

% velocity decrease = (0.362-0.311)/0.362 = 14.1%

EV3:

n(**T_s**) = 0 = - (175/0.43) **(T_s)**+155.6 → T_s = 0.382 N-m

n(**0.1** N-m) = - (175/0.43) **(0.1)**+155.6 = 114.9 rpm

ω(0.1 N-m) = 114.9 rpm * 2π/60 = **12.03 rad/sec**

v = ω · r = 12.03 rad/sec · 0.0254m = **0.306 m/s**

% velocity decrease = (0.357-0.306)/0.357 = 14.3%

The velocity decreases by about 14% for both motors which is similar to the percentage of voltage decrease when dropping from 9 V to 8 V. % Voltage decrease = (9-8)/9 = 11.1%

Section 6.2.3 – 10 Considerations in Motor selection

Before showing some of the details about wound stator DC motors in Section 6.3 and AC motors in Section 6.4, some considerations for motor selection are discussed. Bodine has a motor selection guide that helps with this process [42] and the Micromo DC motor selection guide also provides helpful information [41].

1. The first thing that needs to be determined is the amount of speed and torque that is required. Reference 42 provides guidance for the following 4 types of applications:
    a. Direct drive
    b. Lead screw
    c. Belt dive or rack & pinion
    d. Gear drive, chain & sprocket, or belt & pulley.
2. Next, the type of motor needs to be selected. Table 6.1 shows some pros and cons of different motors. Bodine has a flowchart in the motor selection guide that helps make this decision [42]. The main factor when selecting a motor is often initial cost, but efficiency should also be considered (see Equation 6.18).
3. Should you use gearing and if so what type of gearing should you use? If more torque is needed at the desired speed, gears are likely needed. References 1 and 42 provide some guidance in this area.
4. Voltage supply issues. The type, value, and tolerance of voltage required are important considerations. Is it a fixed voltage or does it fluctuate? Can the effects of the voltage fluctuation be tolerated? For example, if it is a motor that runs on AA batteries, more expensive Lithium Ion batteries might be selected instead of Alkaline batteries if voltage fluctuation (and speed change) can’t be tolerated.
5. Will the motor need to interface with sensors? For example, encoders might be used for speed control.
6. Form Factor and mechanical linkage issues. What diameter and length of shaft is needed? How will the size and weight of the motor affect the application?
7. Temperature of the application and how the motor ratings are affected at different temperatures.
8. Will the motor have a continuous duty cycle or frequently start and stop? If the motor is started and stopped a lot, it is hard on brushes and gears and can lead to them wearing down.
9. Environmental factors. For example, if an explosion is possible with arcing then brushes can’t be used.
10. Maintenance issues. For example, if a motor is difficult to service then brushes are not a good idea.

Section 6.3 – Wound Field DC Motors

As shown in Table 6.1, there are three types of wound field DC motors (Series, Shunt, and Compound). These motors can be used in applications requiring more horsepower than PM-DC motors. These types of DC motors have their magnetic field in the stator produced with a wound field (i.e. electromagnet), which is in contrast to PM-DC motors that have their magnetic field in the stator produced with a permanent magnet. The electromagnet produced by sending current through the stator field windings can produce a much stronger magnetic field than a permanent magnet which results in the capability of producing higher rpm, torque, and horsepower, but on the downside the current flowing through the field wires (that have resistance) results in extra power dissipated by the coil.

Note: *Any time current flows through wire there is an I^2R power loss that results in a reduction of efficiency.*

In a similar way that magnetic wire is wrapped around the rotor poles in Figure 6.8, an electromagnet is created in the stator in wound field DC motors by current flowing through the stator windings. Figure 6.11 shows the three ways that this stator electromagnetic field is created. In this figure, the brushes are colored in solid black and the + and – symbols represent the positive and negative terminals of the supply voltage. The shunt wound motor in Figure 6.11A has the stator field winding (labeled as f) wired in parallel with the armature (or rotor) winding. The series wound motor in Figure 6.11B has the stator field winding wired in series with the armature winding. The motor in Figure 6.11C is called a compound wound motor because it has both a series wound field (labeled as f1) and a shunt wound field (labeled as f2). Reference [11] provides more information about wound stator motors and detailed illustrations that more clearly show the internal wiring.

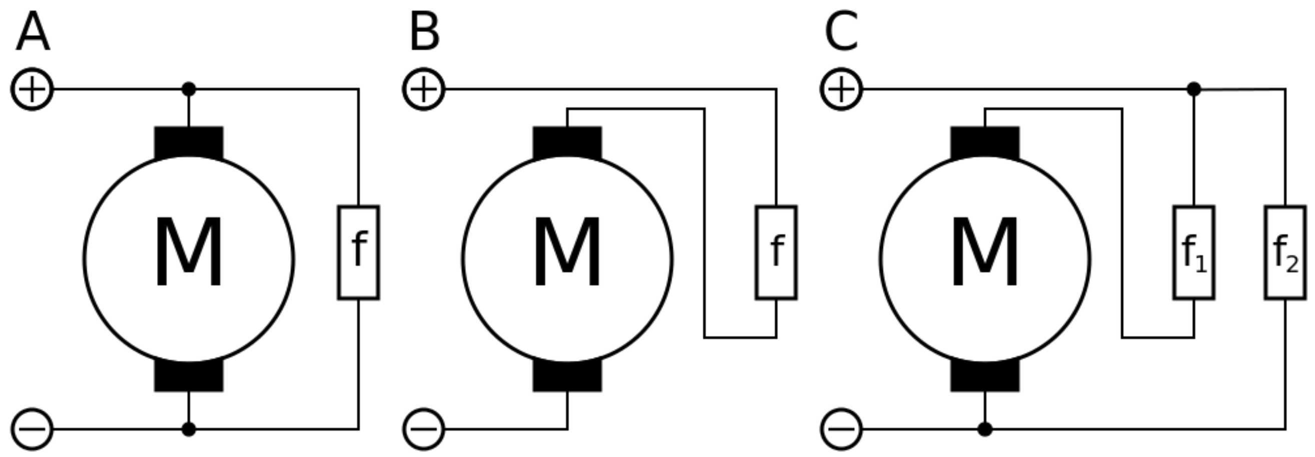


Figure 6.11: (A) Shunt wound – The field is wired in parallel (or Shunt) with the motor, (B) Series wound – The field is wired in series, (C) Compound wound, which is a combination of series and shunt. © Sefid par. Used under CC BY-SA license: https://en.wikipedia.org/wiki/DC_motor#/media/File:Serie_Shunt_Coumpound.svg

Shunt wound motors: Since the field and armature windings are in parallel (i.e. shunt), the current going through them are different, as shown in Figure 6.11A. They operate in the same way as separately-excited DC motors that have separate supply voltages for the stator field and the armature field.

- **Pros:** They have the flattest speed vs torque curve so they have the best speed regulation of any motor and can have their direction easily reversed since the stator field is controlled separately from the armature.
- **Cons:** They have lower rpm and lower starting torque than series wound motors [1]. As mentioned below, they are the most susceptible to motor damage during start up.

Series wound motors: They are often called self-excited because the current going through the stator field winding is the same current that goes through the armature, as shown in Figure 6.11B. Note for Figure 6.12: (A) commutator, (B) brush, (C) rotor (armature) windings, (D, F) stator (field) windings, (E) brush guides.

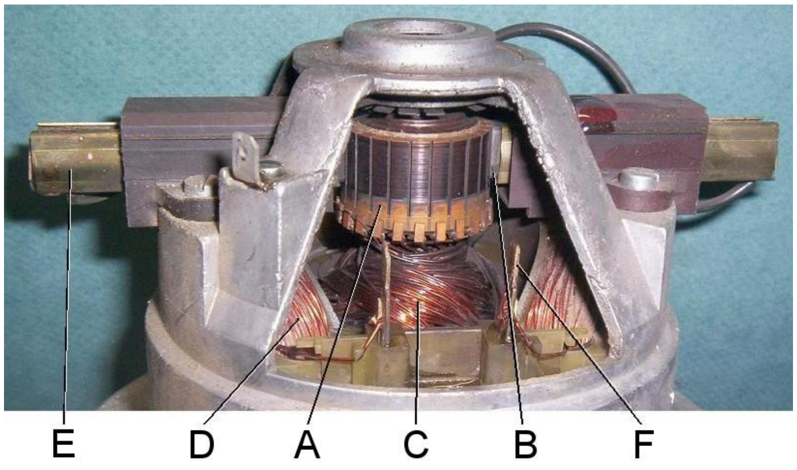


Figure 6.12: Series Wound DC motor internal diagram. Public Domain: https://en.wikipedia.org/wiki/File:Universal_motor_commutator.jpg

- **Pros:** They are also called **Universal Motors** because they can operate on DC or AC supplies. They can operate at high rpm and have the highest horsepower per pound of any motor that can operate from single phase AC power [1].
- **Cons**: In Series wound motors, the speed vs torque curve drops rapidly as torque increases so they are not good at speed regulation. They operate at high rpm, which results in more bearing and brush damage [12].

Compound wound motors: The stator field windings are wired in both shunt and series with the armature windings, as shown in Figure 6.11C. They have a speed vs torque curve that is in between the shunt and series wound DC motors and is closer to the linear speed vs torque in a PM-DC motor.

Motor curves for wound DC motors: The motor curves are much more difficult to create for series, shunt, and compound wound DC motors because they are highly nonlinear. They can be approximated using complicated modeling techniques that are discussed in reference [31], but the modeling techniques are often inaccurate and is outside the scope of this book. Another option is to determine the motor curves experimentally by adjusting the torque to different levels and measuring the current and speed at those torque values in a similar way that was done for the Lego motors in reference [24]. Once enough data (T,n) and (T,i) data points are obtained the speed vs torque and current vs torque can be plotted and an appropriate curve fitting method based on the best fit of the data can be made to create the curves. Then in the same way as was done with PM-DC motors, the output power (Equation 6.4), input power (Equation 6.6), and efficiency (Equation 6.7) can be determined at different torque values that range from 0 to stall torque.

Starting Issues with DC motors: The current in DC motors is approximately equal to: $\mathbf{I = (V_S - emf)/R_M}$, where R_M is the motor resistance, V_S is the supply voltage, and emf is the back emf voltage that is produced by the motor as it spins (Recall the generator action Equation 6.1). Since the back emf is initially zero the full supply voltage is across the motor resistance during the motor starting period and it results in large currents that can damage the motor. The starting current is especially large during startup for shunt motors since the supply voltage is directly across the armature (i.e. $R_M = R_{arm}$) and the resistance of the armature is very small (much less than 1 Ω). Therefore, extremely large currents (V_{sup}/R_{arm}) occur during start-up of a shunt wound DC motor. In order to protect all DC motors (especially shunt wound motors) from excessive currents during start up, an adjustable resistor is often placed in series with the armature. During start up the resistance can be set to a large value to minimize the amount of current flowing into the armature. Once the motor reaches its operating point, the resistance can be reduced. While the motor is operating near its rated load condition, the same variable resistor that was used to safely start the motor can also be used as a mechanism for speed control. Since shunt motors are often used for speed regulation applications, this speed control method is often used. Chapter 9 of reference [31] provides additional information and special circuits that can be used for starting DC motors.

Section 6.4 – AC Motors

Module 5 provided an overview of single phase and three phase AC power, but more details of AC circuits would be useful in understanding AC motors. For that reason, it would be helpful to review the Davis AC Circuits book [14] to provide a deeper understanding of this section. First, it is important to point out that 3-phase and single phase AC motors usually rely on the electrical power grid for input power. Motors are the largest consumer of electrical energy. In order to reduce the energy that is wasted, increasing motor efficiencies for large hp motors is of vital importance. Section 12.60 of NEMA MG 1 [20] shows the efficiency levels required to categorize a poly-phase AC induction motor as a **NEMA Premium** motor. Using a NEMA Premium motor not only plays a big part in reducing energy waste, there is also a relatively short amount of time before the increased initial investment is returned in energy cost savings. This period is called the payback and is calculated in Eq. 6.18 [32].

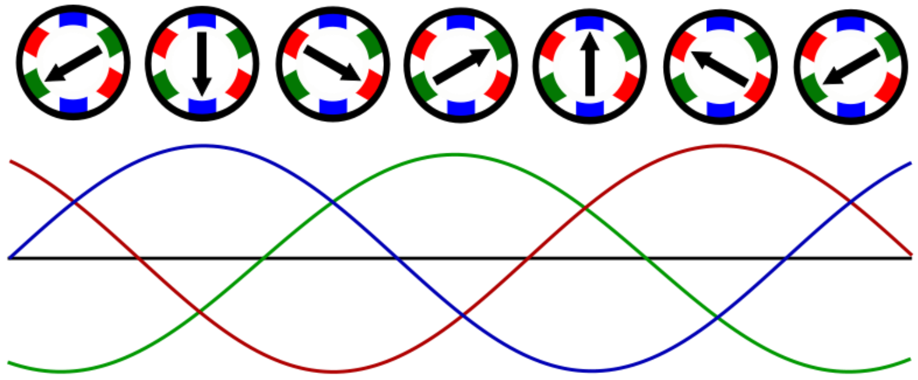
$$[6.18] \quad \text{Payback (years)} = \frac{(\text{Initial Cost Increase for NEMA Premium Motor} - \text{Utlity Rebate})}{\text{Annual Cost Savings}}$$

Note: *There are often Rebates or government programs that offer financial incentives to use less energy.*

Siemens is a major retailer of NEMA Premium motors and has an App that allows the user to "enter a few key data and the app calculates energy efficiency, potential energy cost savings and the pay-back period. These values can be directly compared and the break-even point is displayed graphically. The App also offers an overview of local minimum efficiency requirements for low voltage induction motors (so called MEPS – Minimum Efficiency Performance Standards) for selected countries."[33] Siemens motor homepage also provides numerous resources to assist in motor selection and operation [34].

Section 6.4.1 – Three-Phase AC Motors

AC motors are typically classified as either synchronous or induction motors and usually operate with single phase or 3-phase power (single phase motors are discussed in Section 6.4.2). 3-phase motors work by creating a rotating magnetic field in the stator field windings (as shown in Figure 6.13) that interact with the rotor and cause it to rotate. The red circles in Figure 6.13b represent a squirrel cage rotor (also shown in Figure 6.14), which is the most common type of AC rotor. When clicking on the animation link in Figure 6.13, the stator field is shown to move slightly faster than the red dots that represent the rotor. A synchronous motor will move at the same speed as the stator field (called the synchronous speed, n_S), but an induction motor has **slip**, which results in the rotor moving slower than n_S (see Equation 6.19).


(a)

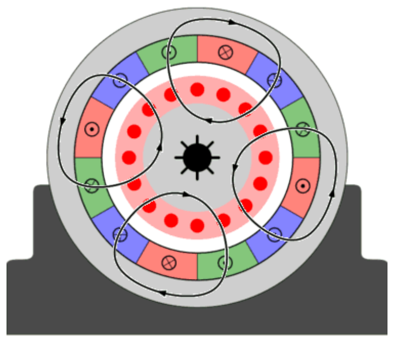
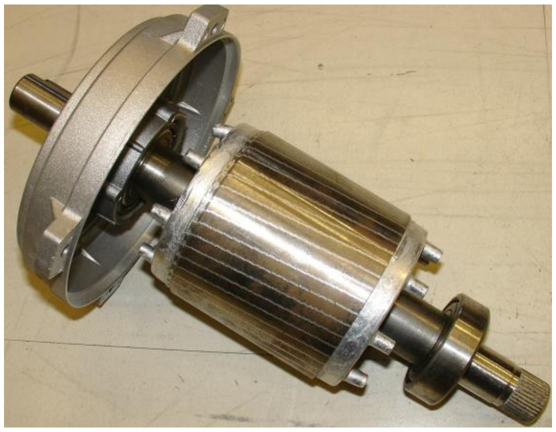
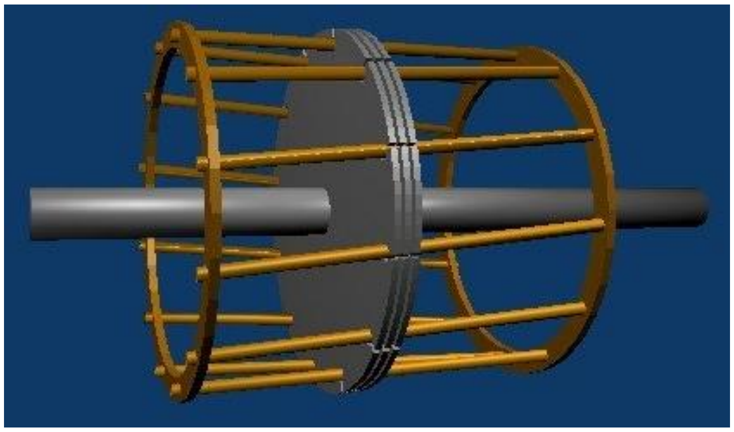

(b)

Figure 6.13: (a) Illustration of the rotating magnetic field in the stator field windings in a 3-phase motor. Public Domain: https://upload.wikimedia.org/wikipedia/commons/8/8c/Rotating-3-phase-magnetic-field.svg **(b) 3-Phase, 4-pole Induction Motor animation (click on the link to show the animation).** © BurnsBurnsBurns. Used under CC BY license: https://en.wikipedia.org/wiki/Induction_motor#/media/File:Asynchronmotor_animation.gif

(a) (b)

Figure 6.14: (a) Photo of a squirrel cage rotor. © Zureks. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Squirrel-cage_rotor#/media/File:Wirnik_by_Zureks.jpg **(b) 3D model of a squirrel cage rotor showing only 3 laminations.** © en:User:Meggar. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Squirrel-cage_rotor#/media/File:Squirrel_cage.jpg

Two important equations for AC motors are shown below [35].

[6.19] $$\mathbf{slip} = \boldsymbol{s} = \frac{\boldsymbol{n_s} - \boldsymbol{n_r}}{\boldsymbol{n_s}}$$

- n_s = synchronous speed of rotor (It is in step with the frequency of the stator rotating magnetic field).
- n_r = actual speed of rotor (It will be less than n_s in an induction motor due to slip, see Figure 6.13b).

[6.20] $$\boldsymbol{synchronous\ speed} = \boldsymbol{n_s} = \frac{\mathbf{120} \cdot \boldsymbol{f}}{\boldsymbol{p}}$$ **Note:** *Eq. 6.20 is also used for generators (see Example 6.10).*

- f = Electric frequency in Hz
- p = Number of poles per phase

Note: *If you have a 3-phase motor with 12 total poles (4 poles per phase), then it is called a 4-pole motor (p = 4).*

Since synchronous motors operate at a constant speed (n_s) without any slip they have higher efficiency and can be used for the highest hp applications of any motors. In a synchronous motor, DC current must be sent through the rotor field coils so it stays lined up with the rotating magnetic field. In contrast, 3-phase induction motors are similar to synchronous motors except they do not require DC current to be sent through the rotor coils and therefore lag behind the speed of the rotating magnetic field (see Equation 6.19). Since 3-Phase induction motors are more widely used, they will be the primary focus of this section. They are typically classified by NEMA as Class A, B, C, or D [31, pg. 422]. Class B are the most commonly used type of 3-Phase AC motors [38]. They have high efficiency, power factor, and starting torque and can start most loads without requiring excessive current. Reference [39] shows hundreds of speed torque curves for 3-phase NEMA Class B induction motors. The speed versus torque curves for AC motors are usually displayed differently than DC motors. Since the starting torque must significantly exceed the Full Load Torque, FLT (i.e. the rated load), the torque is used as the y-axis and put in terms of a percentage of the FLT. Before the motor reaches its FLT speed it must achieve its maximum torque (called the **pullout** or **breakdown** torque). Likewise, the current is also frequently plotted as a percentage of its full load current. Figure 6.15 shows the approximate current and torque versus speed curves for a Siemens, type RGZZESD, 30 hp, 3-Phase, NEMA Class B motor that operates at 460 V, 60Hz, and 1200 RPM [39].

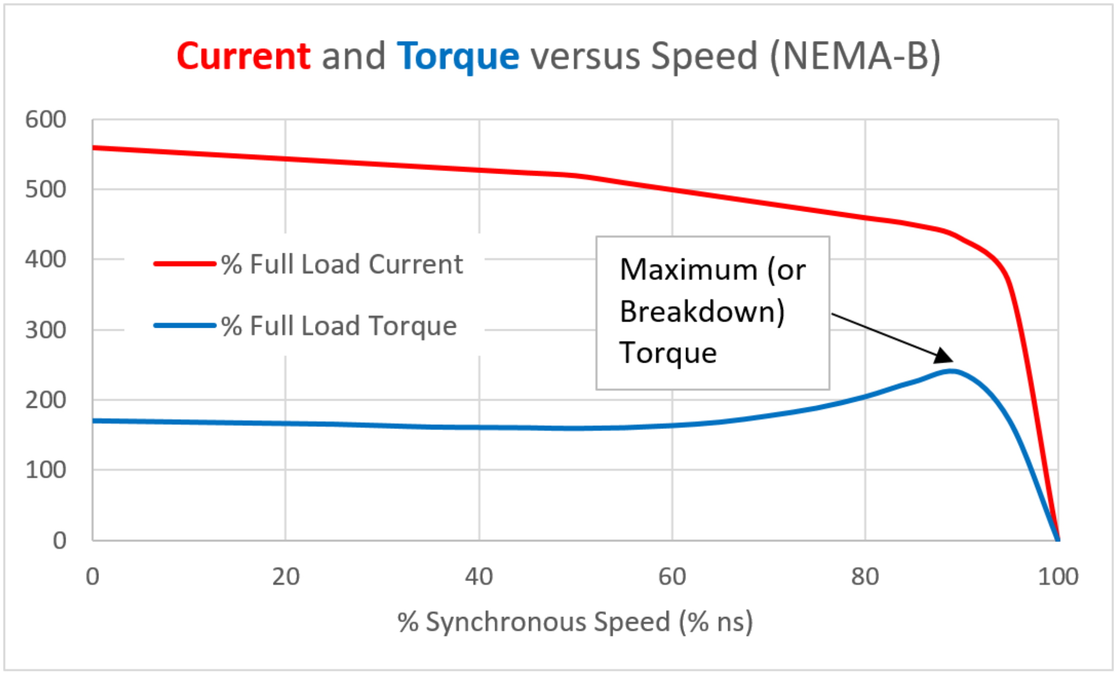


Figure 6.15: % Full Load Current and %FLT versus Speed for a 3-Phase Class B Induction Motor [39].

Section 7.5 of Reference [31] provides more details of the background theory behind the characteristics of the speed versus torque curve for 3-phase AC induction motors and breaks down the relationship in three parts, as shown below.

- **Low-slip region:** This is the nearly linear region on the right side of Figure 6.15 where normal operation of the motor occurs. The synchronous speed (%ns) is close to 100%, which means the slip is very small. According to reference [38], the maximum slip of a Class B motor is typically 5% in this region.
- **Moderate-slip region:** This is middle section of the curve in Figure 6.15 where torque changes drastically and the maximum torque occurs. This maximum torque is called the breakdown or pullout torque and is typically 200% to 250% of the full rated torque for induction motors [31, pg. 405].
- **High-slip region:** This is far left section of the curve in Figure 6.15 that is nearly linear. This is where the motor goes quickly from rest to high speed. 3-phase NEMA class B induction motors typically have a starting torque (i.e. locked rotor torque) equal to 150% to 170% of full load torque (FLT) [38].

Since the current required to start a 3-phase AC motor is much higher than the current required during normal operation (as shown in Figure 6.15) extreme caution must be taken when starting a large horsepower 3-phase AC motor. For example, Dr. Fagan (my mentor professor who is recognized in the preface of the Davis DC Circuits book [13]) tells the story that many years ago the University of Oklahoma Norman campus was once completely blacked out due to the improper starting of a large 3-Phase motor. To avoid stressing the power system or damaging the motor, special starting circuits are often used for 3-phase AC induction motors [31, pg. 432]. While it is risky, 3-phase induction motors are capable of self-starting. However, 3-phase synchronous AC motors always require special starting features or procedures to get them to the synchronous speed. The three most common methods of starting a 3-phase synchronous motor (described in section 6.3 of reference 31) are: 1) Temporarily reducing the speed of the stator field, 2) Using an external prime mover, or 3) Using **Amortisseur** windings that function as a damper during startup. The Amortisseur winding approach is the most commonly used method. According to reference [43], an Amortisseur winding is "a squirrel cage winding placed near the surface of the pole faces of a synchronous motor. Its main purpose is to dampen any speed fluctuations or oscillations that may occur as a result of sudden load changes. It is also used to accelerate the motor during starting. All of the windings are shorted at the two ends of each pole of the motor. As the rotating field moves past the winding it induces currents in the winding which produce torque and accelerate the motor so as to overcome the lag in its speed." The Amortisseur windings are used in an induction motors and are the key to allowing them to work without needing DC current in the rotor field [31, pg. 380].

Power factor (see the Davis AC Circuits book [14] for background) is another important consideration with AC motors. If the power factor is at the maximum value of 1.0, the voltage and current are completely in phase and there is no waste due to reactance in the line. As the inductance and capacitance in the line become more imbalanced, the reactance increases and the voltage and current become more out of phase and the power factor drops. Induction motors result in an inductive (or lagging) power factor. When a large hp 3-phase induction motor is operating under full load torque the lagging power factor typically maxes out at ~ 0.9 [35, 38]. The closer the power factor is to 1.0, the more economical it is to use the motor. As previously mentioned, motors consume a large percentage of the total electrical power used increasing the power factor is important in order to reduce waste as well as save money. Power companies will charge companies based on their cumulative power factor. When an induction motor is started the power factor will be much lower. For example, reference [35] states that it is 0.2 for an induction motor during start up. Synchronous motors have a leading power factor so they can be useful for counteracting the lagging power factor from inductive motor loads [44].

Example 6.9) *A 2-pole 3-phase motor operates using 230 V, 60Hz power. If this motor has a slip of 5% what is the speed of the motor?*

Note: *The rotor of this motor would include 6 total salient poles (2 poles for each phase). A representation of the rotor is shown in Figure 6.16. The field windings would be wrapped around each of the salient poles that are protruding inward. Recall in Figure 6.8 rotors often have salient poles that protrude outward.*

Solution) From Equation 6.20 → ns = 120·f/p = 120 · 60 Hz/2 = 3600 rpm
Be careful: p = 2 (NOT 6) because it is the number of poles per phase.
From Equation 6.19 → slip = $(n_s - n_r)/n_s$ → $n_r = n_s$ (1 – slip) = 3600 (1 – 0.05) = **3420 rpm**

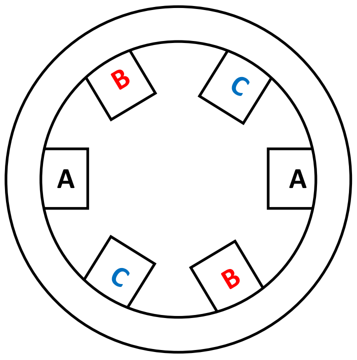


Figure 6.16: Illustration of a 3-phase 2 pole salient pole rotor.

Example 6.10) *The stator for a single phase synchronous generator is constructed with 10 poles (See Figure 6.9b as an example of what the stator might look like). How fast should the rotor be rotated to produce a 60 Hz AC signal?*

Solution) From Equation 6.20 → ns = 120·f/p = 120 · 60 Hz/10 = **720 rpm**

Section 6.4.2 – Single Phase AC Motors

Single-phase induction motors are commonly used for smaller hp applications that require AC power. They are used in similar applications as Universal (i.e. Series Wound DC motors). As compared to a universal motor of similar size, single phase induction motors have higher rated torque and are much better in speed regulation. Single phase induction motors don't require brushes, like Universal motors, so they are quieter and need less maintenance. However, the universal motor has higher rpm and much better starting torque. Since single phase AC motors don't have enough starting torque to get to the operating speed-torque condition they require a starting mechanism. The lack of starting torque occurs because a rotating magnetic field is not created in the stator since only one AC signal is used instead of three. However, once the motor is started and brought near it rated condition it will operate with a speed-torque characteristic that is similar to a 3-phase induction motor in Figure 6.15 (except at much lower horsepower). In order to produce enough torque when starting the motor, an additional winding is often used that is only active during startup. The most common types of single phase induction motors are:

- Split-Phase
- Capacitor Run
- Capacitor Start
- Permanent Split Capacitor
- Shaded Pole

The differences and pros & cons of these 5 types of single phase induction motors are described in detail in references [1] and [45].

One area where single phase induction motors are often used is in air conditioning units. Air conditioner capacitors are one of the most common things to go wrong with a residential AC unit. If your AC compressor motor makes a humming noise, but refuses to start you likely need a new capacitor. Figure 6.17 shows an example of motor starting capacitor.

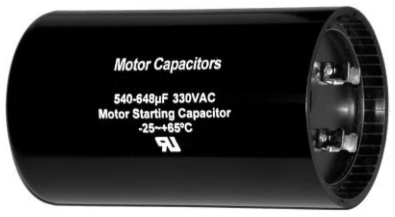


Figure 6.17: Example of a Capacitor used to start a single phase motor. Public Domain: https://en.wikipedia.org/wiki/Motor_capacitor#/media/File:Motor-Start-Capacitor.jpg

A blower fan like the one shown in Figure 6.18 is a common application where a single phase induction motors are used. This blower fan has sharp blades that can severely injure a person if they stuck their hand in the motor while it was running at full speed. However, even with the auxiliary starting mechanism (that are included in all single phase motors) that provides enough torque to initiate rotation, this fan blade can be held with only one finger during start up and that would be enough torque to prevent it from rotating. If this were a 3-phase motor the starting torque is high enough where this would be a very bad idea. This example demonstrates one of the primary difference between single phase and 3-phase motors.

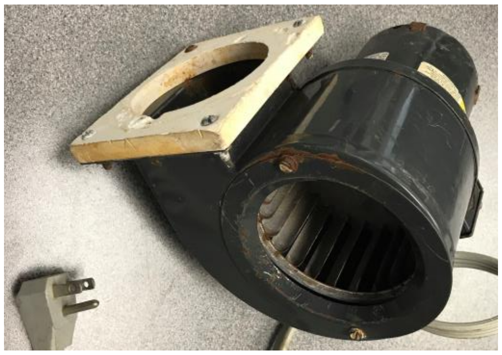


Figure 6.18: Dayton Model 4C446 blower fan that uses a 1/25 hp single phase induction motor.

Module 6 – References and Links

The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[1] Bodine Electric Company, *Bodine Gearmotor Handbook*, 5th ed., 2016. This is a free book that can be downloaded at: http://marketing.bodine-electric.com/acton/media/2588/handbook

[2] DC Motor Overview: https://en.wikipedia.org/wiki/DC_motor

[3] Commutation Overview: https://en.wikipedia.org/wiki/Commutator_(electric)

[4] Overview of electric motors: https://en.wikipedia.org/wiki/Electric_motor

[5] Allan R. Hambley, Electrical Engineering: Principles and Applications 4th, 2007

[6] Brushless DC Motor Overview: https://en.wikipedia.org/wiki/Brushless_DC_electric_motor

[7] Stepper and Servo hp: http://www.machinedesign.com/motorsdrives/problems-horsepower-ratings

[8] Sparkfun Motor Tutorial: https://learn.sparkfun.com/tutorials/motors-and-selecting-the-right-one

[9] Sparkfun Servo Tutorial: https://learn.sparkfun.com/tutorials/hobby-servo-tutorial

[10] Stepper Motor Overview: https://en.wikipedia.org/wiki/Stepper_motor

[11] Types of DC Motors: http://avstop.com/AC/apgeneral/TYPESOFDCMOTORS.html

[12] Series Wound Motor Overview: https://en.wikipedia.org/wiki/Universal_motor

[13] Davis, *DC Circuits*, 2016, https://shareok.org/handle/11244/52245

[14] Davis, *AC Circuits*, 2017, https://shareok.org/handle/11244/51946

[15] Parallax # 900-00008 Continuous Servo Motor datasheet: https://www.parallax.com/sites/default/files/downloads/900-00008-Continuous-Rotation-Servo-Documentation-v2.2.pdf

[16] Parallax # 900-00005 180^o Servo Motor datasheet: https://www.parallax.com/sites/default/files/downloads/900-00005-Standard-Servo-Product-Documentation-v2.2.pdf

[17] RF-370CA Mabuchi PM-DC Motor: https://www.jameco.com/Jameco/Products/ProdDS/238473.PDF

[18] Jameco MD5-1885 motor: *https://www.jameco.com/webapp/wcs/stores/servlet/Product_10001_10001_232022_-1*

[19] Miller, T., Mcgilp, M., Staton, D., & Bremner, J. (1999). Calculation of inductance in permanent-magnet DC motors. *IEE Proceedings - Electric Power Applications,146*(2), 129-137.

[20] National Electrical Manufacturing Association, *NEMA MG 1 Motors and Generators Standards Document*, 2009. https://law.resource.org/pub/us/cfr/ibr/005/nema.mg-1.2009.pdf

[21] American National Standards Institute, Changes to NEMA MG 1-2016 – Motors and Generators, https://blog.ansi.org/2016/10/nema-mg-1-2016-motors-generators/#gref

[22] National Electrical Manufacturing Association, *NEMA MG 1 Motors and Generators* Standards Document, http://www.nema.org/Standards/Pages/Motors-and-Generators.aspx

[23] Dynamics Overview: https://www.engineeringtoolbox.com/dynamics-t_60.html

[24] Lego NXT and EV3 Motor Details: http://philohome.com/motors/motorcomp.htm

[25] 555 Timer Overview: https://en.wikipedia.org/wiki/555_timer_IC

[26] 555 Timer Experiments: https://www.allaboutcircuits.com/textbook/experiments/chpt-8/555-ic/

[27] LM555 Datasheet: http://pdf.datasheetcatalog.com/datasheet/nationalsemiconductor/DS007851.PDF

[28] Pololu 5:1 Micro Metal Gear Motor Datasheet, https://www.pololu.com/product/1000

[29] Pololu 10:1 Micro Metal Gear Motor Datasheet, https://www.pololu.com/product/999/specs

[30] Sveum, *FE Exam Review: Electrical and Computer Engineering*, PPI, 2006. ISBN: 978-1-59126-069-1.

[31] Chapman, *Electric Machinery Fundamentals*, 4th ed. McGraw-Hill, 2005. ISBN: 978-0-07-246523-9.

[32] Motor Challenge Fact Sheet – Buying an Energy-Efficient Electric Motor, US Department of Energy, https://www.energy.gov/sites/prod/files/2014/04/f15/mc-0382.pdf

[33] Siemens App that includes an energy savings calculator: https://www.industry.siemens.com/drives/global/en/motor/low-voltage-motor/energy-savings-calculator/Pages/Default.aspx

[34] Siemens Motors: https://www.industry.siemens.com/drives/global/en/motor/Pages/Default.aspx

[35] Induction Motor Overview: https://en.wikipedia.org/wiki/Induction_motor

[36] PM-DC Motor speed versus torque curves: http://lancet.mit.edu/motors/motors3.html

[37] All About Circuits Introduction to AC Motors: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/introduction-ac-motors/

[38] All About Circuits Induction Motors: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/tesla-polyphase-induction-motors/

[39] Speed Torque Curves for hundreds of 3-Phase Motors. https://www.industry.usa.siemens.com/drives/us/en/electric-motor/nema-motors/Literature-and-technical-resources/Documents/app-man-section5-part2-speed-torque-curves.pdf

[40] Wildi, *Electric Machines, Drives, and Power Systems*, 5th ed. Pearson Ed., 2002. ISBN: 0-13-093083-0.

[41] Micromo DC Motor Guide: https://www.micromo.com/technical-library/dc-motor-tutorials

[42] Bodine Motor Selection Guide. http://www.bodine-electric.com/asp/putstream.asp?context=/pdf/bodine_123_selection_guide.pdf

[43] Amortisseur windings definition: https://en.wiktionary.org/wiki/amortisseur_winding

[44] All About Circuits Synchronous Motors: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/synchronous-motors/

[45] All About Circuits Single Phase Motors: https://www.allaboutcircuits.com/textbook/alternating-current/chpt-13/single-phase-induction-motors/

Module 7 – Computer Communications

This module provides an overview of the operation principles and common types of computer communication. Numerous references are also included in this module in case more details are needed. The overarching goal is to provide enough background and details that the reader would be equipped to connect their computer to a device with a serial interface and communicate with it using LabVIEW. Once the knowledge to do serial communication is gained, it is a very easy step to do other types of communication.

Note: *At first glance, this final module might not seem as important as some of the others, but from my engineering consulting experience being able to communicate with instruments and devices has been one of the most beneficial skills to have. In order to automate any electromechanical system or process you need to get all of the data to the CPU. This can be done by either using a DAQ device (as discussed in Module 3) or an instrument that has a built-in communication interface (Table 7.1 shows examples). Most engineers desire to automate their systems or processes and often have to hire a consultant or find someone in their organization that specializes in that area to do it for them. I hope that this book empowers individuals of all types of backgrounds to do this type of work themselves.*

In Figure 1.4, the ASCII table was provided and ASCII character coding was discussed. After using ASCII to code an alphanumeric message with ones and zeros, the next logical step is to send the ASCII data to a computer or a peripheral device. The idea of sharing ASCII or other forms of data between computers or devices is often referred to as computer communication or data communication. There are many different types of **protocols** (i.e. communication standards) that are deployed to so that computers/devices can transmit data to each other. The protocols are the set of rules that are used to assure that the transmitter and receiver will know how to communicate with each other [1, 2]. Many forms of computer communication can be categorized as either parallel or serial. Figure 7.1 shows that parallel communication separates each bit of data on different lines and transmits them in parallel, but serial communication sends all of the data together in series in one bit stream.

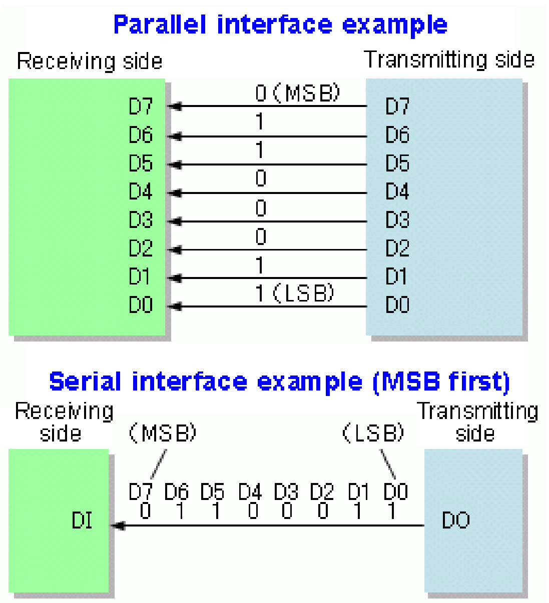


Figure 7.1: Parallel and serial communication bit transmission illustration. © SyamilAshri. Used under CC BY-SA license: https://en.wikipedia.org/wiki/Serial_communication#/media/File:Parallel_and_Serial_Transmission.gif

Most forms of parallel communication allow 8 bits to be transmitted at once, which made it a very important interface in the past when high speed data transmission was more difficult to achieve. It became a very popular form of communication in application that required transmission of large amounts of data (such as printers) over short distances. As speeds increased and the ability to buffer data and perform other specialized hardware functions became easier to achieve, **serial communication** became much more commonly used than **parallel communication** [3, 4]. Since serial allows all data to be transmitted bit by bit on 1 wire it is much more cost effective when data is transmitted long distances.

Note: *Figure 7.2 shows the RS-232 serial bit-stream that includes signaling bits that are used to help the sender and receiver communicate.*

There are many types of communication protocols commonly used today and many classic ones that had widespread use in the past. Table 7.1 shows a list of popular types of computer communication.

Table 7.1) Popular Types of Computer Communication.

Motor Type	Type	Summary and Notes
LPT (IEEE-1284)	Parallel	The "parallel port" was the standard used for almost all printers before USB [33].
GPIB (IEEE-488)	Parallel	**G**eneral **P**urpose **I**nterface **B**us is a more advanced form of the parallel port [5].
RS-232	Serial	**R**ecommended **S**tandard 232 was the first widespread form of short length (usually 15m or less) asynchronous serial communication [13, 24]. The serial port is controlled by a **U**niversal **S**ynchronous **R**eceiver **T**ransmitter (UART) IC [37].
RS-422	Serial	4-wire version of RS-232. Transmits data up to 1500 meters or at 10 Mbps [15].
RS-485	Serial	Similar to RS-422, but allows to connect up to 32 devices (multi-point) [16].
USB	Serial	**U**niversal **S**erial **B**us continually updates to higher speed standards [7, 8].
I^2C	Serial	**I**nter-**I**ntegrated **C**ircuit – Form of synchronous serial communication [9, 10].
Bluetooth	Serial	**U**ltra-**H**igh **F**requency (UHF) wireless form of short range communication [11,12].

There are also a variety of different forms of communication protocols used in networking applications such as Ethernet, 10-BaseT, TCP/IP, etc. [6, 34]. Many instruments can be communicated with using Ethernet/TCP/IP protocols. For example the Fluke 8845A/8846A Multimeter product class cab be communicated with using GPIB, RS-232, or Ethernet [29]. In LabVIEW, many of these protocols can be communicated with using NI-VISA. The term VISA stands for **V**irtual **I**nstrument **S**oftware **A**rchitecture. It is an API that is used to "communicate with most instrumentation buses including GPIB, USB, Serial, and Ethernet. It provides a consistent and easy to use command set to communicate with a variety of instruments."[27]

RS-232 [13] has been an industry standard that engineers have used to communicate with devices for decades and will be the primary focus of this module, but the information provided will also be applicable to many other forms of communication. RS-232 is a form of serial communication (i.e. it transmits bits in series) that has one transmit line and one receive line (i.e. 2 total wires used for data). A nice succinct one page overview of the RS-232 specification is shown in [14]. There are also 4-wire serial communication standards called RS-422 [15] and RS-485 [16]. These 4-wire standards allow for greater transmission lengths because they are more immune to noise since they transmit and receive "differentially" on two wires (i.e. 4 total wires used for data). By transmitting or receiving on two wires, the noise that is common to both wires can be removed.

Note on 2-wire versus 4-wire: *The 2-wire RS-232 serial protocol is an example of single-ended data transmission and the 4-wire RS-422 or RS-485 serial protocols are examples of double-ended (or differential) data transmission. This is an important distinction in many different types of communications and even in other fields, such as electronics. Operational Amplifiers are a good example of an integrated circuit that can be operated in a single ended configuration when noise is not a concern, but operated in differential configuration when it is desired to cancel out the signal that is* ***common*** *to both input lines. Noise is a "common" signal on the two input lines and it is usually unwanted so removing it using the differential configuration is a frequently used practice. For most types of differential transmission, there is a limit to how large the common signal can be before it cannot be effectively canceled out. This limit is often referred to as the Common Mode Rejection Ratio (CMMR) [17]. Reference [18] includes a video produced by Texas Instruments that explains CMRR, as well as other applicable noise related parameters such as Signal to Noise Ratio (SNR) and Total Harmonic Distortion (THD).*

The length of data transmission and noise immunity are important aspects of communication, but the synchronization is another important distinction between types of communication. RS-232 is an example of **asynchronous communication**, where the data are NOT transmitted by a clock signal, as done in **synchronous**

communication. Without a clock, special signaling schemes (i.e. start and stop bits) are used to help the sender and receiver communicate with each other. This adds additional complexity and the overhead by adding extra bit results in less data transmitted in a given time (i.e. less throughput) [32]. Figure 7.2 shows how the bits are ordered in RS-232 transmission. This is called asynchronous communication since there is no master clock that governs the transmission. During the idle times between transmissions the line is held at a digital logic high level (Binary 1). The voltage levels required by the RS-232 standard are shown below [14]. The logic is an active low style, where a binary 0 (i.e. space) results in a positive voltage and a binary 1 (i.e. mark) results in a negative voltage:

- Binary 0 (called a space): Transmitted signal +5 to +15 V_{DC}, Received Signal: +3 to +13 V_{DC}
- Binary 1 (called a mark): Transmitted signal -5 to -15 V_{DC}, Received Signal: -3 to -13 V_{DC}

The RS-232 levels are typically at levels of ± 13V and often need to be converted to TTL Levels (0V to 5V). There are adapters available capable of this task and also integrated circuits that can also accomplish this in hardware.

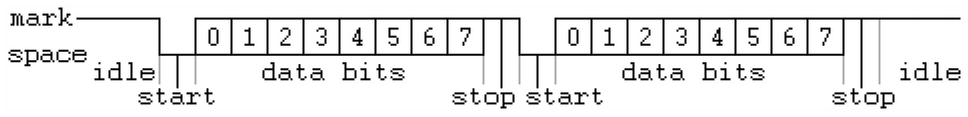


Figure 7.2: RS-232 transmission protocol. Public Domain: https://en.m.wikipedia.org/wiki/Asynchronous_serial_communication#/media/File%3APuerto_serie_RS-232.png

In addition to the **start and stop signaling bits** shown in Figure 7.2, a **parity bit** that is frequently added to the RS-232 bit stream. Parity is a rudimentary form of error checking that is frequently used in RS-232 and other forms of communication. As demonstrated in examples 7.2 through 7.4, parity checking can be used to catch an odd number of bit errors in a stream of data bits, but cannot catch an even number of bit errors. The downside of including a parity bit is that there is additional overhead to the data transmission process [21, 22, 36].

Example 7.1) *If there are 8 data bits, 1 start bit, 2 stop bits, and 1 parity bit then each time 8 bits of data (i.e. 1 byte) were transmitted how much longer would it take? This can also be called the %overhead.*

Solution) The overhead includes 1 start bit, 2 stop bits, and 1 parity bit to have a total of 4 bits of overhead.

12 bits are transmitted instead of 8 so the transmission time increases by (12-8)/8 = 50%. **%overhead = 50%**.

If a parity bit is allocated and used for error checking, then either odd or even parity is used to set the parity bit:

- **Odd** – The parity bit is set so that the total number of ones in the data bits & parity bit is odd.
- **Even** – The parity bit is set so that the total number of ones in the data bits & parity bit is even.

If a parity bit is allocated, but NOT used for error checking, then either mark or space is used to set the parity bit:

- **Mark** – The parity bit is always set to 1.
- **Space** – The parity bit is always set to 0.

Example 7.2) *If the following bytes of information (given in hexadecimal) are transmitted 8 bits at a time and odd parity is used, determine the value of the parity bit for each byte. Data → E3 9A*

Solution)
E3 = 1110 0011 → There are 5 ones in E3. Since it is already an odd number, the **parity bit is 0**.
9A = 1001 1010 → There are 4 ones in 9A. Since it is an even number of ones, the **parity bit is 1** to make it an odd number of ones.

Example 7.3) *If the following bytes of information (given in hexadecimal) are transmitted 8 bits at a time and even parity is used, determine the value of the parity bit for each byte. Data → FD 36*

Solution)
FD = 1111 1101 → There are 7 ones in FD. Since it is an odd number of ones, the **parity bit is 1** to make it an even number of ones.
36 = 0011 0110 → There are 4 ones in 36. Since it is already an even number, the **parity bit is 0**.

Note: *Figure 7.2 shows the number of data bits is 8 (bits 0 to 7), but depending on the application it could be set to any number of data bits. For example, a commonly used RS-232 transmission type is sending ASCII symbols that only require 7 bits (see the ASCII Table in Figure 1.4). With only 7 bits needed in this application, the parity bit could be added and the total number of bits for data and parity would fit into one byte (i.e. 8 bits). Example 7.4 demonstrates this type of transmission.*

Example 7.4) *If 9 bits of data with the following pattern 101010101 need to be transmitted, but a UART is used that only allows 8 data bits. How can the UART be used to transmit all 9 bits? No parity checking is needed.*

Solution)
Since UART ICs have a parity bit and no parity is needed for this transmission the parity bit can be used to transmit the 9th bit. Mark parity will be used so that the parity bit is set to 1, which will be used to transmit the most significant bit and the 8 least significant bits will be transmitted normally. Once received the parity bit can be add as the MSB to the 8 data bits that are received.

Example 7.5) *ASCII characters 7Up are transmitted with RS-232 using 7 data bits,* ***even*** *parity, 1 start bit, and 1 stop bits.* ***A)*** *Write the bit stream out assuming the data moves from right to left from the sender to receiver with the form: [Start Bit][Parity Bit][Data Bits][Stop Bits]* ***B)*** *Calculate the %overhead for this transmission?*

Solution)
A) From Figure 1.4, 7Up = $(37\ 55\ 70)_{16}$ → $(37)_{16}$ = $(\mathbf{011\ 0111})_2$ $(55)_{16}$ = $(\mathbf{101\ 0101})_2$ $(70)_{16}$ = $(\mathbf{111\ 0000})_2$

7 → [0][**1**][**011 0111**][1] * There were 5 ones in the data bit so the parity bit is **1** for even parity.

U → [0][**0**][**101 0101**][1] * There were 4 ones in the data bit so the parity bit is **0** for even parity.

p → [0][**1**][**111 0000**][1] * There were 3 ones in the data bit so the parity bit is **1** for even parity.

B) There are 3 overhead bits and 7 data bits so the % overhead = (10-7)/7 = **42.9%**

Example 7.6) *The ASCII character E is transmitted with RS-232 using 7 data bits,* ***odd*** *parity, 1 start bit, and 2 stop bits. A) Write the bit stream out assuming the data moves from right to left from the sender to receiver with the form: [Start Bit][Parity Bit][Data Bits][Stop Bits] B) Also, calculate the %overhead for this transmission? C) If the middle bit of the 7 bit ASCII character is flipped during transmission will the parity error checking detect the bit error? D) If the first and last bit of the 7 bit ASCII character is flipped during transmission will the parity error checking detect the bit error?*

Solution)
A) From Figure 1.4, E = $(45)_{16}$ $(\mathbf{100\ 0101})_2$

E → [0][**0**][**100 0101**][11] * There were 3 ones in the data bit so the parity bit is **0** for odd parity.

B) There are 4 overhead bits and 7 data bits so the % overhead = (11-7)/7 = **57.1%**

C) If the **middle bit** is flipped the receiver receives 100 **1**101. The parity bit would be determined by the receiver to be 1 because there are 4 ones in the data bits. Since the parity bits from the sender and receiver don't agree then the receiver detects an error in the transmission.

D) If the **first and last bits** are flipped the receiver receives **0**00 010**0**. The parity bit would be determined by the receiver to be 0 because there are 1 one in the data bits. Since the parity bits from the sender and receiver agree then the receiver doesn't detect an error in the transmission.

Example 7.6 shows that when an even number of bit errors occurs, parity checking will not detect the error. For this reason, parity checking is only good for certain situation, like detecting issues with transmission cables. There are many more sophisticated error detection algorithms that can be used to detect any type of bit error. Two of the more popular methods are [26]:

- **Hamming Codes** add additional overhead by using multiple parity bits, but can detect all bit errors and in some situations correct bit errors. It must be implement on a fixed number of data bits.
- Cyclic Redundancy Check (CRC) runs a block of data of any size through a mathematical algorithm called a polynomial and the **result** of the polynomial (sometimes referred to as a **checksum**) is transmitted along with the data. When the receiver gets the data, it can be sent through the same polynomial to see if the same results are obtained. With CRC no error correction can be made.

While error detection is an important aspect in data communication, the transmission speed is of even greater concern. The speed at which the bits are transmitted is referred to as **Baud rate** and is usually given units of bits per second (**bps**). The term Baud rate and bits per second (bps) are often used interchangeably, but technically they are different according to [19, 20]. The faster the Baud rate is set, the higher the likelihood of bit errors. Therefore, it is best to set the Baud rate to a bps value as low as possible while still transmitting the data within the desired time window. The following examples (7.7, 7.8, 7.9, and 7.10) provide more details on Baud rate.

Example 7.7) *The phrase "Hello World" was used in Example 1.2 to show how the ASCII table can be used to assign hexadecimal values to ASCII symbols. If the hexadecimal data that corresponds to "Hello World" is transmitted at a Baud rate of 9600 bps with no parity, 7 data bits, 1 start bit, and 1 stop bit how long will it take to transmit the data and what is the %overhead?*

Solution) There are 13 ASCII symbols that are transferred with 7 bits each, which makes 91 total data bits. For each of the 13 bit frames there are 2 overhead bits, makes a total of 26 bits.

- %overhead = 2/7 = 26/91 = **28.6%**
- Time = (91 bits)/(9600 bits/sec) = .00948 seconds = **9.48 ms**

When considering transmission speed, it is important to understand the definition of the terms kilobyte (kB), Megabyte (MB), Gigabyte (GB), and Terabyte (TB). Usually, the standard prefixes aren't used when stating a number of Bytes. For example, 1kB is usually referred to as 1,024 Bytes instead of 1,000 Bytes. While the format shown in the bullets below aren't always followed, they are considered by most to be the standard definitions.

- 1 kB = 2e10 Bytes = 1,024 Bytes
- 1 MB = 2e20 Bytes = 1,048,576 Bytes
- 1 GB = 2e30 Bytes = 1,073,741,824 Bytes
- 1 TB = 2e40 Bytes = 1,099,511,627,776 Bytes

To make things even more confusing, when using a prefix with the bits per second (bps) Baud rate parameter, standard prefix values are used. For example, when the internet was in its early stages, 14.4 kbps serial modems were the most popular devices used to login to the internet through a service provider such as AOL. A 14.4 kbps modem transmitted 14,400 bits per second, which is 14.4 * 1000 instead of 14.4 * 1024. Currently, there are many UART chips (the IC that is used for RS-232 communication) that have a maximum Baud rate of 115.2 kbps. A list of different UART types are shown at reference [37].

Example 7.8) *Calculate the %overhead and The number of minutes it takes to transmit 5 MB of data using the following RS-232 communication parameters.*

- *Baud Rate = 115.2 kbps or 115,200 bps (**Note:** 1 kbps = 1000 bps)*
- *Data Bits = 5*
- *Parity = odd*
- *Start Bits = 1 (**Note:** you always have 1 start bit)*
- *Stop Bits = 2*

Solution) Each data frame has 5 data bits and 4 overhead bits.

- %overhead = 4/5 = **80%**

This situation (with 5 data bits and 4 overhead bits) is usually the worst case scenario when using RS-232 communication with parity error checking.

Total number of data bits = 5MB · (2e20 Bytes/MB) · (8 Bits/Byte) = 41,943,040 bits

Taking into account the overhead bits the total number of bits to be transmitted

- Total bits = 41,943,040 bits · (1 + %overhead) = 41,943,040 bits · 1.8 = **75,497,472 bits**
- Total time = 75,497,472 bits/115,200 bits per second = 655.36 seconds = 655.36/60 = **10.92 minutes**

Many RS-232 UARTs allow the Baud rate to be adjusted by the user. This allows the user to select the value of the Baud rate that allows the data to be transmitted within the user's desired timeframe. When adjustments can be made it is important to set the Baud rate as low as possible in order to reduce the likelihood of bit errors. A practical example of this occurred recently when trying to read data fast enough from serial GPS receivers in order to produce the differential GPS corrections that would be used by airplanes to navigate more accurately. The following two examples shows the thought process that was used in order to select the Baud rate and message size for a GPS application.

Example 7.9) *A GPS receiver that allows multiple log files to be transmitted via RS-232 communication was used. It was determined that when using a Baud rate of 115.2 kbps errors occurred, so the next highest rate in the adjustable Baud rate device was used, which was 57.6 kbps. There are 10 different log files that can either be enabled or disabled for transmission and all of the log files include approximately 1 kB of data. If you needed to receive updates of the GPS log files every 0.565 seconds and you had a 25% overhead needed for bit framing (i.e. the start and stop bit, 2/8), how many log files could be enabled for transmission?*

Solution) The amount of bits that can be transmitted in 0.565 seconds using a Baud rate of 57,600 bps is 0.565 seconds * 57,600 bits/second = 32,544 data bits

- Total bits = 32,544*(1 + 0.25) = 40,680/8 = 5,085 Bytes → 5,085/1024 = **4.97 kB**
- **Four log files** could be enabled.

If a fifth log file was enabled it would take slightly longer than 0.565 seconds. In a situation like this the "floor" of the number must be taken when it is converted to an integer instead of rounding it to the nearest integer.

Example 7.10) *A GPS receiver equipped with a serial communication interface allowed for one of the following Baud rates to be selected: 14.4 kbps, 19.2 kbps, 38.4 kbps, 57.6 kbps, and 115.2 kbps. If you needed to receive 10,000 total bits (including data bits and overhead bits) in less than 0.25 seconds in order for the differential GPS application software to work properly, which Baud rate would be the best to select.*

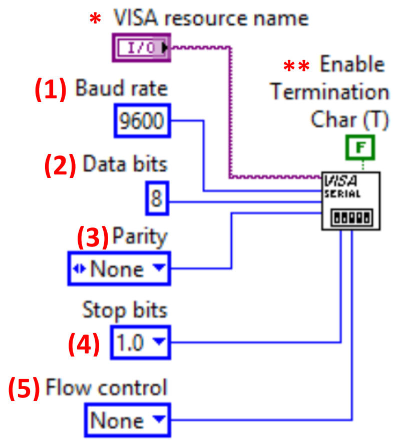
Solution)

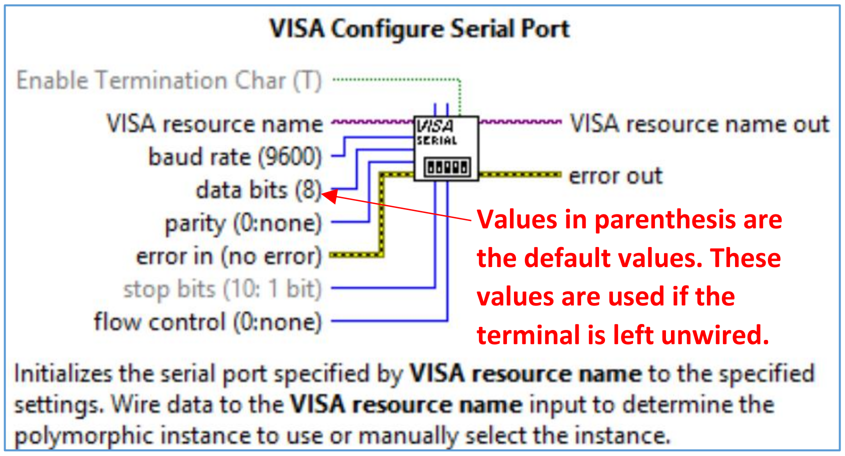
- minimum Baud rate needed to transmit 10,000 bits in 0.25 seconds is 10,000/0.25 = 40,000 bps = 40 kbps

This means that either the 57.6 kbps or the 115.2 kbs settings are fast enough to work. Since the 115.2 kbps would be more likely to result in bit errors (as mentioned in the previous example) it would be best to select the **57.6 kbps**.

When implementing serial communication in LabVIEW the previously mentioned items of **Baud rate**, **parity type** and number of **data bits** and **stop bits** must be selected along with a parameter called **Flow Control**.

Note: *There are **5 steps** for RS-232 communication. Step 1 is configuring the **5 parameters** shown in Figure 7.3a.*





Initializes the serial port specified by **VISA resource name** to the specified settings. Wire data to the **VISA resource name** input to determine the polymorphic instance to use or manually select the instance.

Figure 7.3: (a) Configuring a Serial Port in LabVIEW, (b) LabVIEW Context Help for VISA Configure Serial Port.

(1) **Baud Rate** is a metric for transmission speed and is given in units of bits per second (**bps**). The constant entered for Baud rate in Figure 7.3a is the default value of 9600 bps, but the baud rate input terminal in the VISA Configure Serial Port VI will accept any value that is entered.

(2) **Data Bits** – Figure 7.2 shows the number of data bits is 8 (bits 0 to 7), which is the default number of data bits value used in LabVIEW.

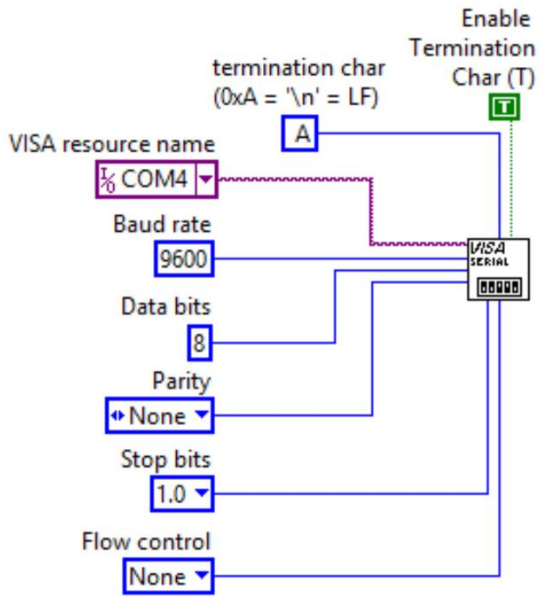
(3) **Parity** – The default setting in LabVIEW is for no parity to be used (i.e. None), but if parity is implemented then usually Odd parity or Even parity are used, but there is also the option of mark or space.

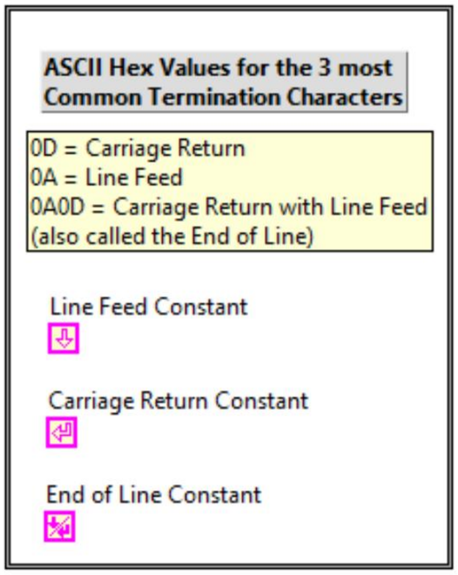
(4) **Stop Bits** – As shown in Figure 7.2 the logic level returns to a logic high (1) after the data is transmitted. The default is 1 stop bit, but LabVIEW allows 1.5 and 2 as other options. Start bits are also used, but there is always one start bit so there is no reason for it to be configurable. Together, the Start and Stop bits are often called "framing bits" that add additional overhead to the data transmission.

(5) **Flow Control** (Also called handshaking) – RS-232 can employ a back and forth protocol to govern the data transfer process. The most common method is the Ready to Send (RTS), Clear to Send (CTS) flow control method. In RTS/CTS, the sender asserts the RTS line low to tell the receiver it is ready to send the data, and then the receiver asserts the CTS line low to tell the sender it is ready to receive the data [21, 23, 24].

* The **VISA Resource Name** in Figure 7.3a is the LabVIEW control where the serial port is selected. Once the serial driver is installed for the hardware then a new "COM" number will appear. A good way to do a test to make sure the correct COM port is selected and working properly is by doing a loopback test (See Example 7.11).

** The Enable Termination Character is set to False in Figure 7.3a. If it is set to True, an ASCII character can be selected that sets the termination of a data packet. Line Feed, Carriage Return, and Carriage Return with Line Feed are the most popular types of termination characters and can be wired into the terminal to the left of the Enable Termination Character terminal as shown in Figure 7.4. I usually prefer to set the Enable Termination Character to False in the configuration VI (as shown in Figure 7.3) and manually add a termination character at the end of the data packet that is to be transmitted (see Examples 7.1 and 7.2).
The VISA resource name is set as a constant instead of a control in Figure 7.4. Setting it as a constant, makes the COM port number hard-coded on the block diagram instead of something that is selected on the front panel. The VISA resource name constant and the parity, Stop bits, and Flow control constants all have an arrow on their right side that allows the user to select the desired value from a list options instead of just typing in a value. This type of constant is called an enumeration (or enum) constant.





ASCII Hex Values for the 3 most Common Termination Characters

0D = Carriage Return
0A = Line Feed
0A0D = Carriage Return with Line Feed (also called the End of Line)

(a) (b)

Figure 7.4: (a) Configuration with Termination Character Enabled. (b) 3 Most Common Termination Characters

Other than the 5 parameters highlighted in Figure 7.3, the "endianness" is another consideration. **Endianness** refers to the byte order. When words with more than one byte are transmitted, the order in which the bytes are ordered must be selected. In big-endian format the most significant bytes of the word are ordered earlier than the least significant bytes and in little endian the least significant bytes are ordered earlier [25].

In order to connect a computer to a serial instrument and communicate with it, the following steps must be taken to connect the instrument to the computer.

1) Install serial port hardware on the computer. Since serial ports are rarely available on computers today the easiest way to do this is to use a USB to serial adapter, as show in Figure 7.5a.
2) Buy or make a loopback plug and create a program to test to see if the port is functioning and if you can send data out of pin 3 and receive it back on pin 2
   Note: *The loopback plug simply ties pins 2 and 3 together so a homemade version of it can easily be created*).

3) Determine if a "Null Modem" is needed [28] (See Figure 7.5e). If both the sender and receiver have their ports in the same order, then the wires need to be crossed so that one device can talk to the other device. For example, a Null Modem cable is needed to connect the Fluke 45 (See Figure 7.5d) to a computer because otherwise both devices have the same ports and the TX line will collide with the other TX line.
4) Determine if any other adapters are needed. For example, The Yokogawa UT-350 PID Controller needs a RS-422 to RS-232 adapter. That adapter has a DB25 plug on it so it needs a DB25 to DB9 adapter so that it can connect to the USB to RS-232 adapter (See Figure 7.5f).

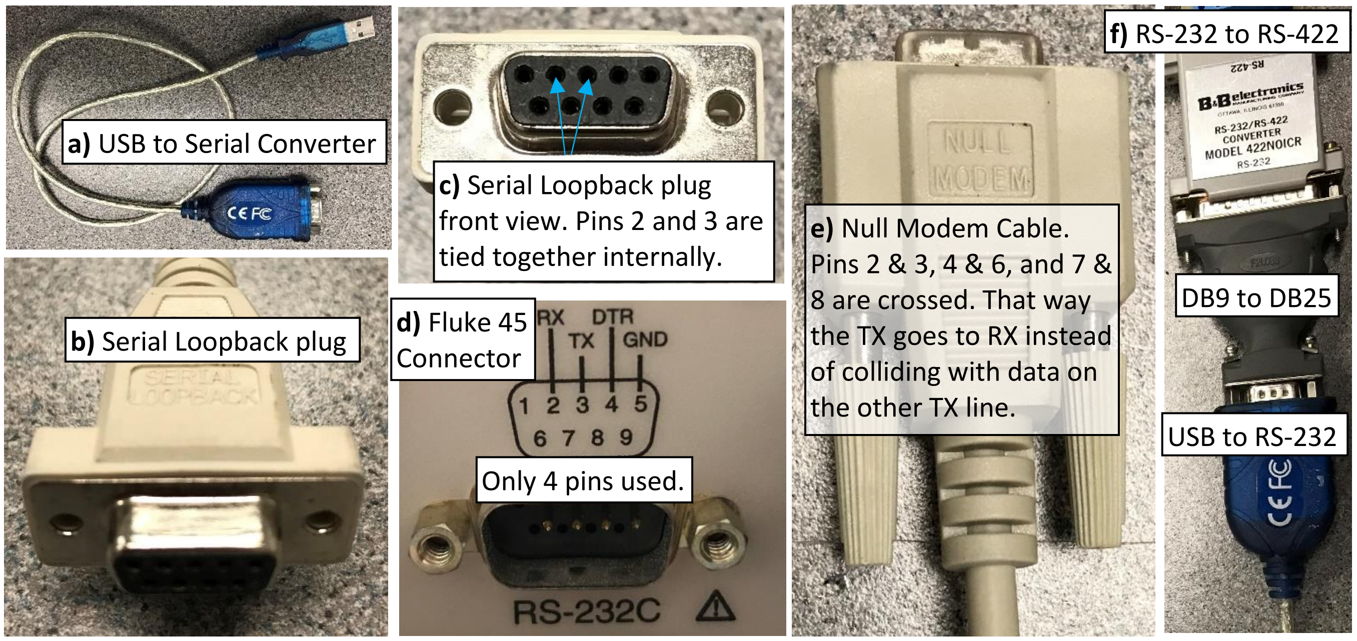

a) USB to Serial Converter
b) Serial Loopback plug
SERIAL LOOPBACK
c) Serial Loopback plug front view. Pins 2 and 3 are tied together internally.
d) Fluke 45 Connector
RX DTR
TX GND
1 2 3 4 5
6 7 8 9
Only 4 pins used.
RS-232C
e) Null Modem Cable. Pins 2 & 3, 4 & 6, and 7 & 8 are crossed. That way the TX goes to RX instead of colliding with data on the other TX line.
NULL MODEM
f) RS-232 to RS-422
B&B electronics
RS-232/RS-422 CONVERTER MODEL 422NOICR
DB9 to DB25
USB to RS-232

Figure 7.5: a) USB to RS-232 Serial Converter, b) Loopback serial plug, c) DB9 female plug, d) DB9 male connector on a Fluke 45 Multimeter (Only 4 pins are used), d) Null Modem Crossover Cable, f) Serial Adapters.

After connecting to the device with a properly functioning serial port the following steps must be taken to communicate with the device in LabVIEW (or another program such as Matlab [21]).

1) **Initialization** – Make sure the 5 parameters in Figure 7.3 are set to the same parameters on both devices.
2) **Write** a command – The programming guide of the device will provide the message structure for writing.
3) **Wait** – There is a delay in how long it takes the device to respond (i.e. Baud rate). Wait extra time to be safe.
4) **Read** from the device – The programming guide will provide the message structure of the received data.
5) **Parse** – The received data isn't very useful without using the LabVIEW string palette to parse the message.

This 5-step process is demonstrated in the following examples: Example 7.11 – Loopback test, Example 7.12 – Receiving data from a Fluke 45 Multimeter, Example 7.13 – Sending & receiving data from a Yokogawa UT350. By showing a few examples that are completely different, it should help with setting up serial communication for other devices that you come across.

Example 7.11) *Create a LabVIEW program to perform a loopback test.*

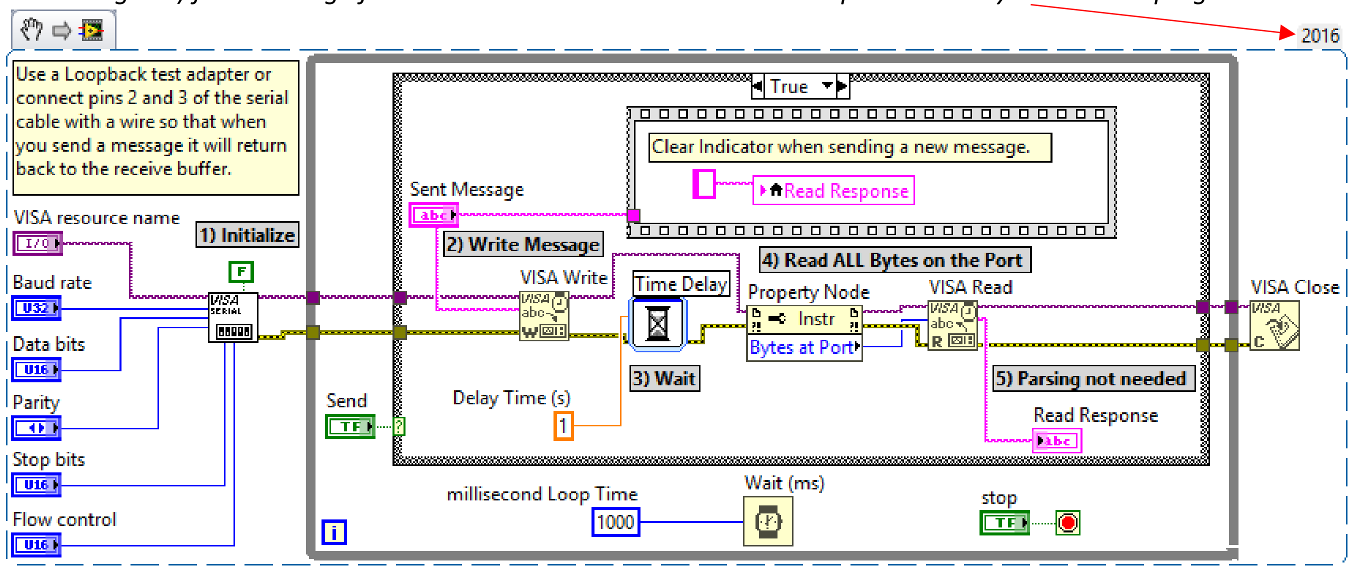
When installing the serial drivers it is a good idea to test the port with a serial loopback cable (as shown in Figure 7.5b). Even if the serial port has been installed and working it is a good troubleshooting technique to use a loopback test to verify that data can be transmitted and received. Figure 7.6 shows a **snippet** (See note below about Snippets) of the LabVIEW Block Diagram that will perform a loopback test. In this example, there is a True/False case structure. When the Send button is pressed the True case is selected and the data that is typed in the Sent Message control is sent through the loopback plug and then received in the serial read buffer. When you execute this program and press the Send button, the ASCII characters that are entered into the Sent Message control will be sent out the transmit line and the signal will travel through the serial loopback plug and then be received in the Read line. If there are no issues with the serial interface and the plug is connected, then the characters in the Sent Message control will appear in the Receive Message indicator. When the Send button is pressed, the Read Response indicator is emptied so that it is a blank field. If the Read Response indicator remains blank after the send button is pressed, then it indicates that there is a problem with the serial interface or the loopback cable was not connected. When the Sent button is not pressed, the false case is executed which is an idle state. All the false case does is pass the purple VISA resource name wire and error wire through to the right side of the case structure so that if the program is stopped the VISA close VI will function properly. It is a good idea to always close the VISA port AFTER all communication is completed (not after each time a message is read) because otherwise the port will stay open and it might cause you problems when trying to communicate with it later. To quickly find the VIs for steps 1 to 4 shown below press control-space and type the following:

1) VISA Configure Serial Port **2) VISA Write** **3) Time Delay** **4) VISA Read**

To find the other VIs (or blocks) in LabVIEW right click and search through the palettes or do the following:

- For the VISA close VI, just press control-space and type: **VISA Close**
- Wait (ms) sets how often the while loop is called. To find it press control-space and type: **Wait (ms)**
- The Property Node that is labeled in Blue Font "Bytes at Port" sends an output of the total number of bytes that are in the read buffer. It is usually best to read all of the bytes at the read buffer so it will be empty for the next iteration. To find this VI, press control-space and type: **VISA Bytes at Serial Port**

Note: *A* ***Snippet*** *can be created in LabVIEW by selecting the parts of the program that you want to save and then selecting the* ***Edit*** *Menu and selecting "****Create VI Snippet from Selection****." The Snippet file will be saved as a .png image file that can be viewed as if it were a screenshot, but it can also be imported (or just dragged back into the Block Diagram) for the image file to become LabVIEW code that corresponds to the year in the top right corner.*


Use a Loopback test adapter or connect pins 2 and 3 of the serial cable with a wire so that when you send a message it will return back to the receive buffer.

Clear Indicator when sending a new message.

Figure 7.6: Snippet of the Block Diagram for the Serial Loopback Test.

Example 7.12) *Create a LabVIEW program to read the numbers on the display of a Fluke 45 Multimeter.*

Before creating the LabVIEW program the following steps need to be completed.

1) **Get the Programmers Manual** for the device you want to communicate with and determine the communication parameters. In this case, a very common RS-232 configuration is used, which is 9600 bps, 8 data bits, No Parity, 1 stop bit [29]. The Fluke 45 is an older Fluke model, but it uses the same RS-232 commands as newer models (Fluke 8845A/8846A) so this manual will be used. Finding the exact manual is usually not necessary. As long as it is in the same family of products, the programming guide will likely be the same. The one big difference in the newer Fluke model is that it allows the device to be connected as either a RS-232, GPIB, or Ethernet device and the menu structure on the Fluke allows the type to be set.
2) **Connecting to the Fluke** - As previously mentioned the Fluke 45 requires the use of a Null Modem to connect to a computer. The USB to RS-232 adapter (Figure 7.5a) is connected to the Null Modem cable (Figure 7.5e) and that cable is plugged into the RS-232 DB9 port on the back of the Fluke 45 (Figure 7.5d).
3) **Set up the Fluke** 45 for RS-232 communication in the menu on the front panel (See the left instrument in Figure 7.7). Since the Fluke 45 only does RS-232 communication it just needs to be enabled and the Baud Rate set using the bottom right button.
4) **Find the command** needed to write to the Fluke to read the display (meas1? Followed by a carriage return) and the message structure of the returned message to determine how to parse it and convert it from a string to a number. The message structure of the returned message is shown in Figure 7.9.

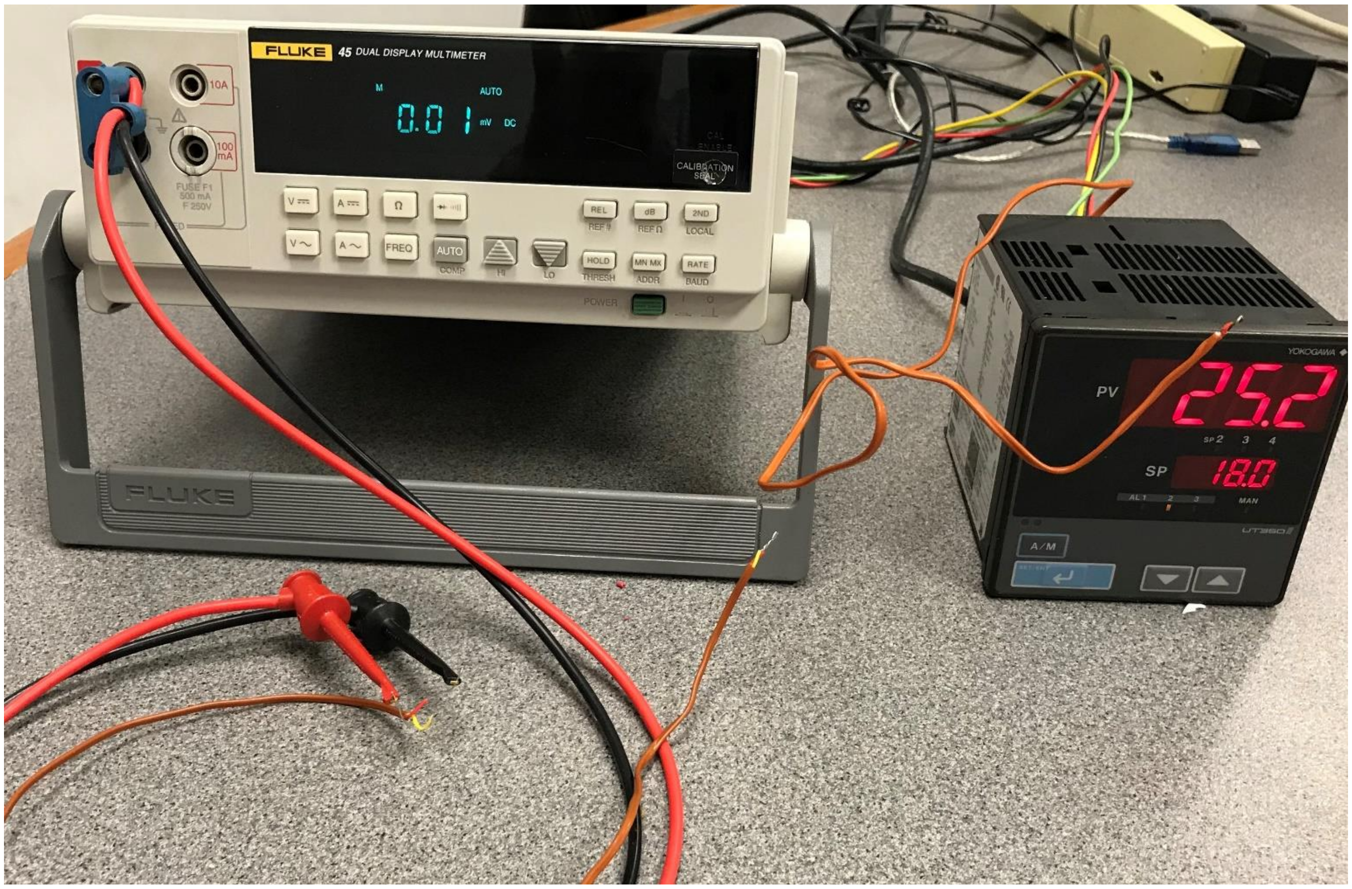


Figure 7.7: (Left) The Fluke 45 set up to read a K-Type Thermocouple (Red and Yellow wires), (Right) Yokogawa UT-350 set up to read a J-Type Thermocouple (Red and White wires) [30]. The brown shield of the thermocouple wires indicate they are "Thermocouple Grade" instead of "Extension Grade."

Now the Fluke 45 is ready to be communicated with, as shown in the snippet of the Block Diagram in Figure 7.8. As was the case in the loopback VI, when the case is False the program is idle.

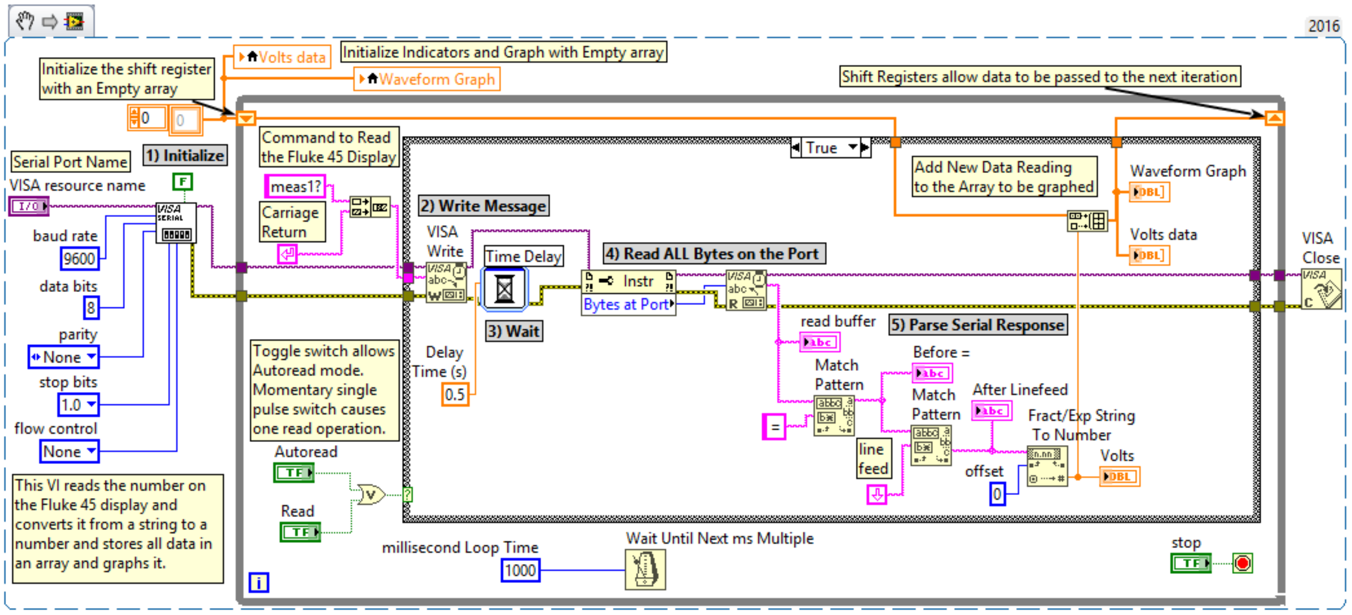
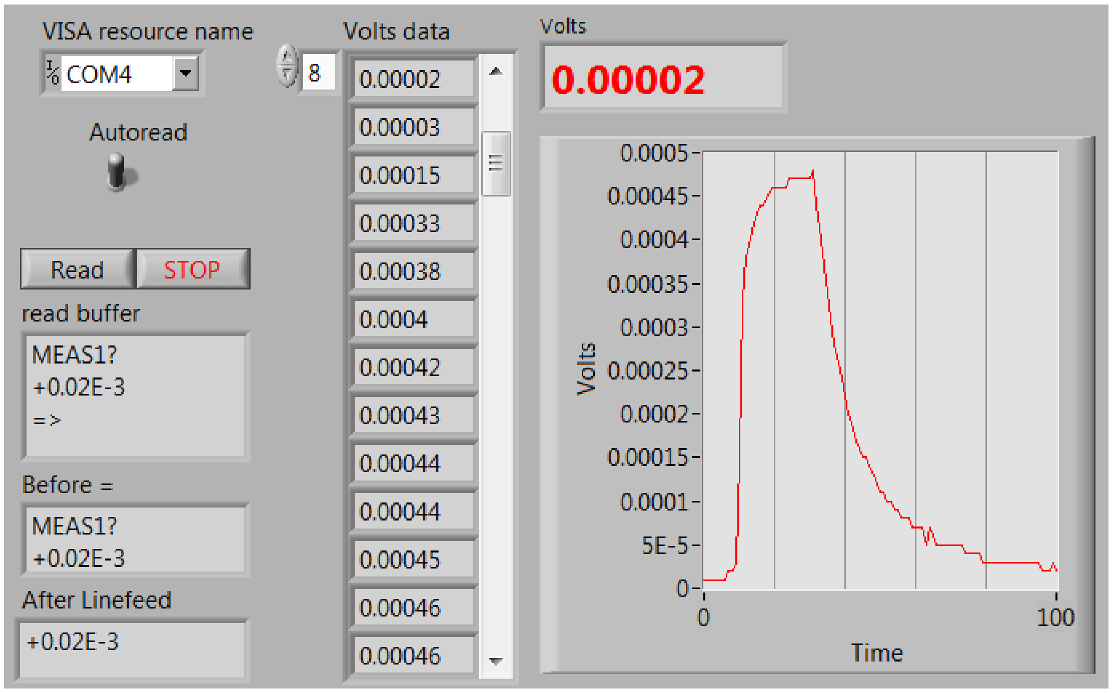

Initialize the shift register with an Empty array
Volts data
Initialize Indicators and Graph with Empty array
Waveform Graph
Shift Registers allow data to be passed to the next iteration
Serial Port Name
1) Initialize
VISA resource name
Command to Read the Fluke 45 Display
meas1?
Carriage Return
True
Add New Data Reading to the Array to be graphed
Waveform Graph
Volts data
VISA Close
baud rate
9600
data bits
8
parity
None
stop bits
1.0
flow control
None
2) Write Message
VISA Write
Time Delay
3) Wait
4) Read ALL Bytes on the Port
Instr
Bytes at Port
read buffer
5) Parse Serial Response
Before =
Match Pattern
Match Pattern
After Linefeed
Fract/Exp String To Number
Volts
offset
0
line feed
Delay Time (s)
0.5
Toggle switch allows Autoread mode. Momentary single pulse switch causes one read operation.
Autoread
Read
This VI reads the number on the Fluke 45 display and converts it from a string to a number and stores all data in an array and graphs it.
millisecond Loop Time
1000
Wait Until Next ms Multiple
stop
2016

Figure 7.8: Snippet of the Block Diagram for the Fluke 45 RS-232 program to read the value on the display.

The following figure shows 100 voltage readings (1 per second) taken from a K-Type thermocouple when going from room temperature (~ 24 ^{o}C) to temperature when holding the thermocouple between 2 fingers (~ 36 ^{o}C). The voltage difference is ~ 0.47 mV. Table 4.2 showed that the sensitivity of a K-Type thermocouple is 41 µV/^{O}C. The ΔV of 0.47 mV corresponds to a ΔT of ~ 11.5 ^{o}C, which is very close to the estimated ΔT of 36 ^{o}C - 24 ^{o}C. The read buffer shows the raw set of characters received and the two boxes below shows how parsing was used.



read buffer
MEAS1?
+0.02E-3
=>

Before =
MEAS1?
+0.02E-3

After Linefeed
+0.02E-3

Volts data:

```
0.00002
0.00003
0.00015
0.00033
0.00038
0.0004
0.00042
0.00043
0.00044
0.00044
0.00045
0.00046
0.00046
```

Volts: 0.00002

Figure 7.9: Front Panel of the Fluke 45 program showing the reading of a J-Type Thermocouple.

Example 7.13) *Create a LabVIEW program to read and write to a Yokogawa UT-350 PID Controller.*

Before creating the LabVIEW program, the 4 steps below need to be completed.

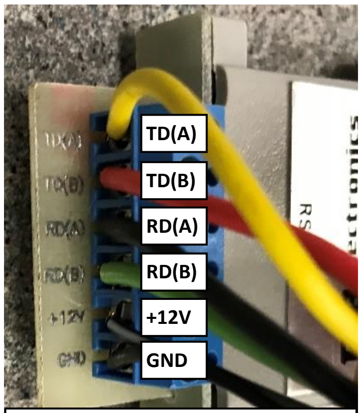


Figure 7.10: Wires connecting the Yokogawa UT-350 to the RS-422 to RS-232 Adapter

1) Get the Programmers Manual for the device you want to communicate with. The Yokogawa UT-350 is discontinued, but the programmer's manual is still on their website along with other specification documents [31].
2) Connecting to the Yokogawa – This device doesn't require a Null Modem. However, as shown in (Figure 7.5f) adapters are needed to convert from RS-422 to RS-232 in order to use the USB to RS-232 adapter that was used in the previous examples to connect it to a computer. If the computer needs to be farther from the Yokogawa device a straight through DB9 serial cable can be used to connect between the USB to RS-232 converter and the DB9 to DB25 adapter (. Figure 7.10 shows a closer look at the 2 transmit lines (TD(A) and TD(B)) and the two receive lines (RD(A) and RD(B)) that make up this 4 wire device. Figure 7.11 shows the connections of these wires made to the Yokogawa and also a description of all of the terminals that can be used, including the thermocouple input. This device actually uses RS-485 protocol instead of RS-422, but the RS-422 to RS-232 adapter also worked. The RS-485 standard allows multiple units to share the data bus (in this case up to 31 Yokogawa UT-350 units) [31].
3) Set up the Yokogawa (See the right instrument in Figure 7.7). There are dip switches in the unit that allows the serial communication parameters to be changed: Data bits: 7 or 8, Parity: Yes or No, Parity bit type: Odd or Even, Stop Bit Selection: 1 or 2 bits, and to change the Baud rate between 300, 600, 2400, 4800, 9800, or 19200 bps.
4) Find the commands needed to read the display and write out the set point to the Yokogawa and determine the message structure of the returned message to determine how to parse it and convert it from a string to a number so it can be plotted. The messaging structure for this device is incredibly difficult and took a lot of trial and error before finally getting it to work. The message structure of the read and write messages are shown in the Block Diagram snippets in Figures 7.14 and 7.16. The returned message for each case is shown in Figures 7.15 and 7.17.

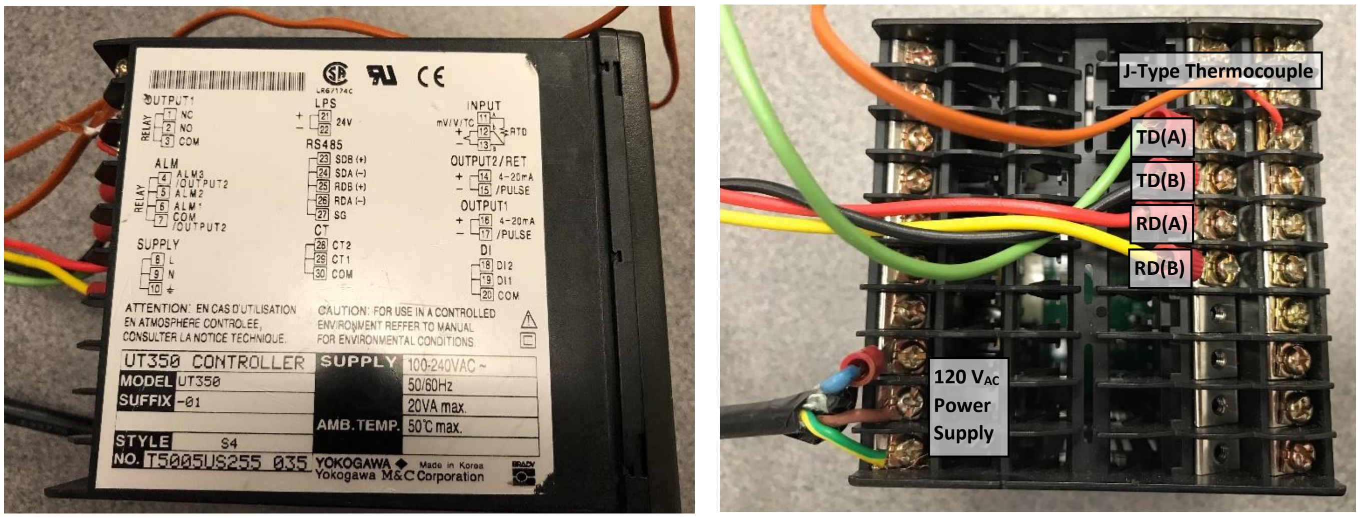


Figure 7.11: Yokogawa UT-350 Controller Wiring Backplane and Labels.

Like in the previous two examples, a case structure is used, but in this program there is no idle state. When the Send Set Point button (i.e. momentary switch) is False the program continuously reads the temperature value on the UT-350 display every cycle. Each cycle is hard-coded to 1 second by wiring 1000 ms into the Wait (ms) block. The slow sample rate is acceptable in this cause because speed is not important for this temperature monitoring application. If the Send Set Point button is pressed, the True case is selected (Shown in Figures 7.15 and 7.17) and the temperature is not read, but instead the Set Point is transmitted to the Yokogawa. Since it is a momentary switch (i.e. It has Latch Mechanical Action in LabVIEW) it goes right back to False after one cycle. This program was much more difficult to implement than the Fluke 45 due to the complicated message format.

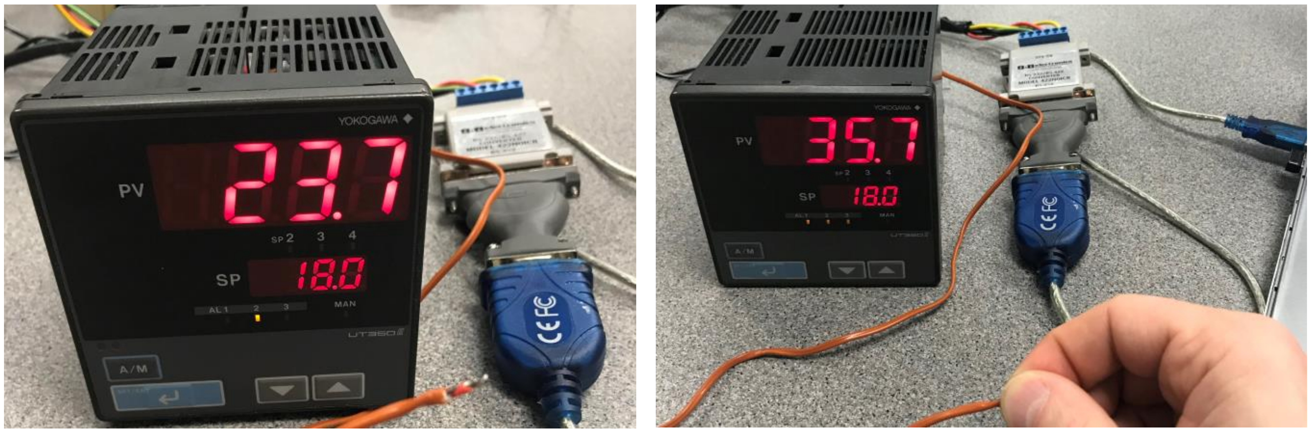


Figure 7.12: Yokogawa UT-350 (a) Thermocouple at room temperature, (b) Thermocouple body temperature

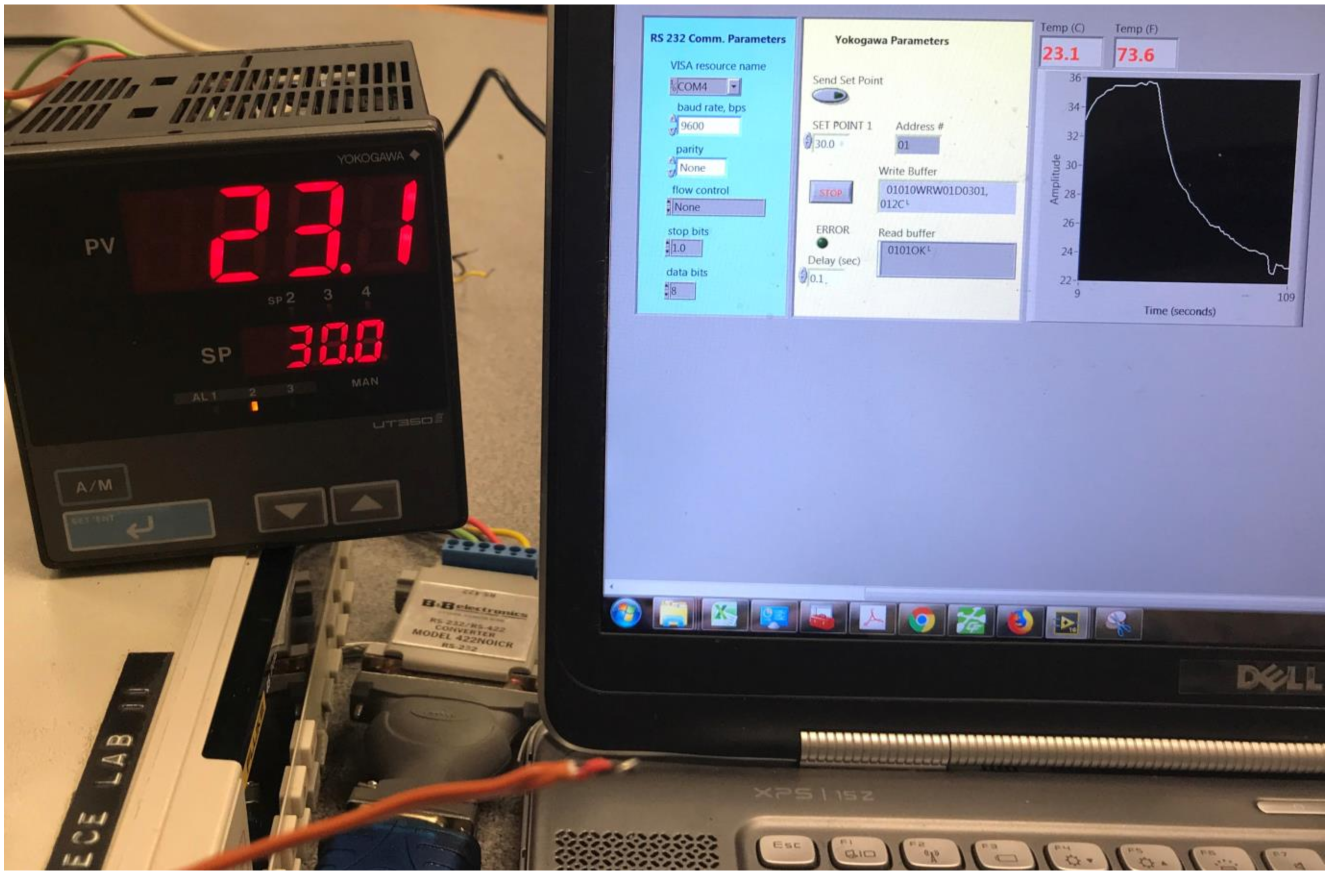
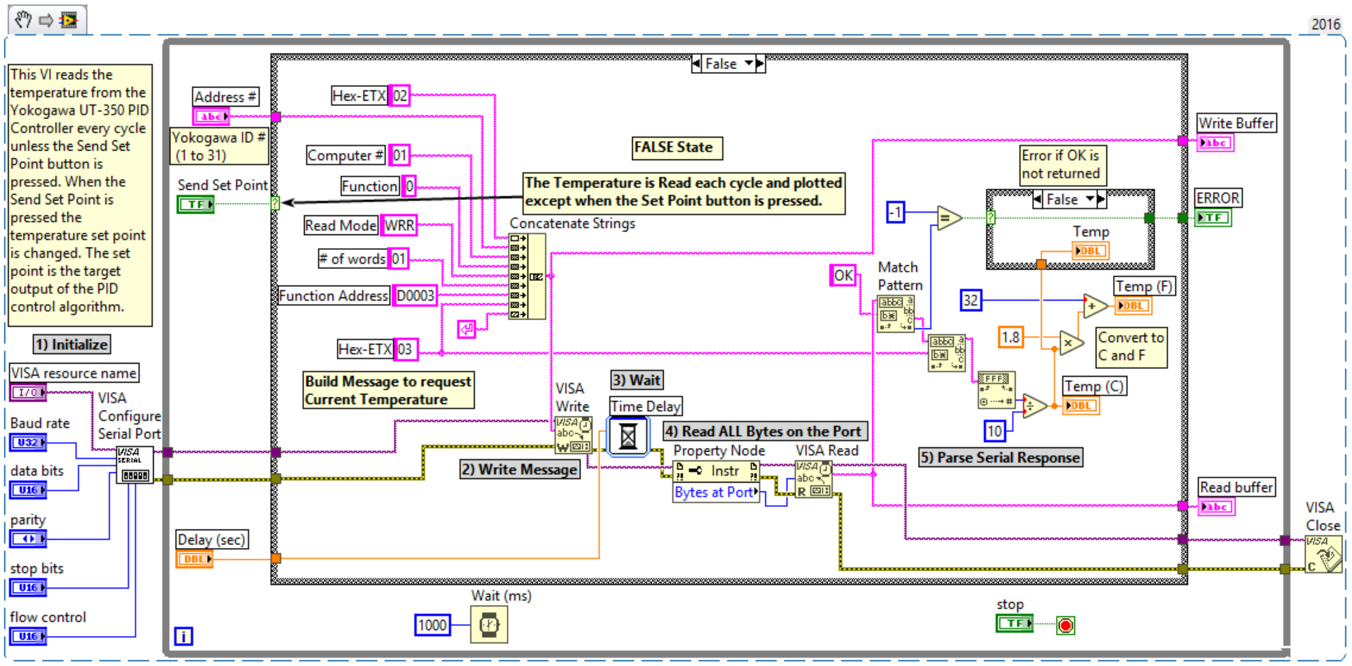


Figure 7.13: Yokogawa UT-350 showing that the set point is changed from 18 ^oC (in Figure 7.12) to 30 ^oC.

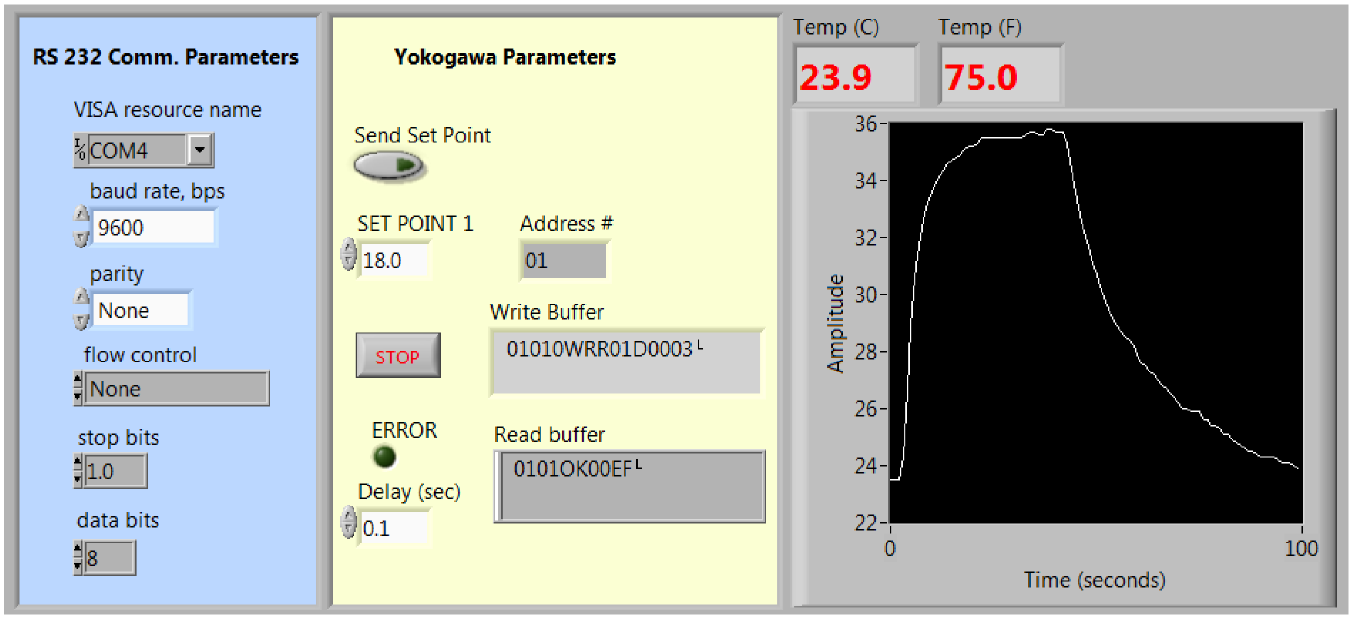

This VI reads the temperature from the Yokogawa UT-350 PID Controller every cycle unless the Send Set Point button is pressed. When the Send Set Point is pressed the temperature set point is changed. The set point is the target output of the PID control algorithm.

FALSE State

The Temperature is Read each cycle and plotted except when the Set Point button is pressed.

Build Message to request Current Temperature

Figure 7.14: Snippet of Yokogawa UT-350 Controller program to read the Thermocouple value.


RS 232 Comm. Parameters

VISA resource name: COM4

baud rate, bps: 9600

parity: None

flow control: None

stop bits: 1.0

data bits: 8

Yokogawa Parameters

Send Set Point

SET POINT 1: 18.0

Address #: 01

Write Buffer: 01010WRR01D0003

Read buffer: 0101OK00EF

ERROR

Delay (sec): 0.1

Temp (C): 23.9

Temp (F): 75.0

Figure 7.15: Front Panel of Yokogawa UT-350 Controller reading the Thermocouple value.

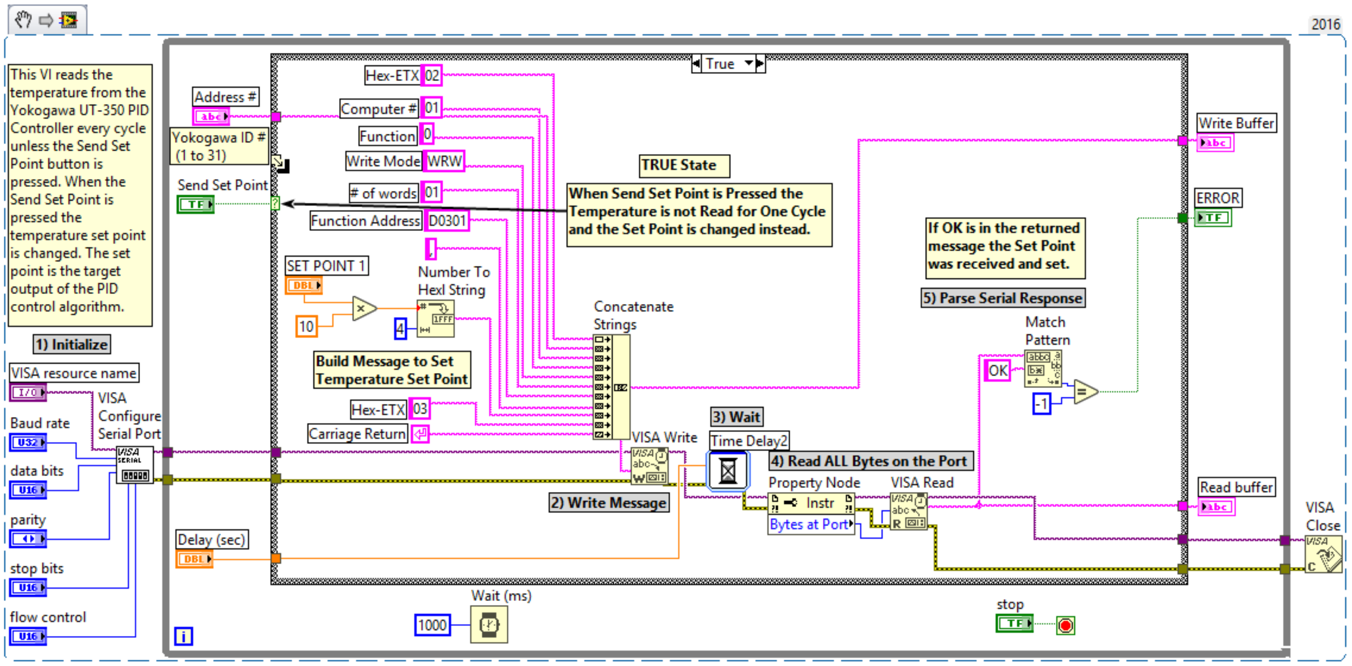
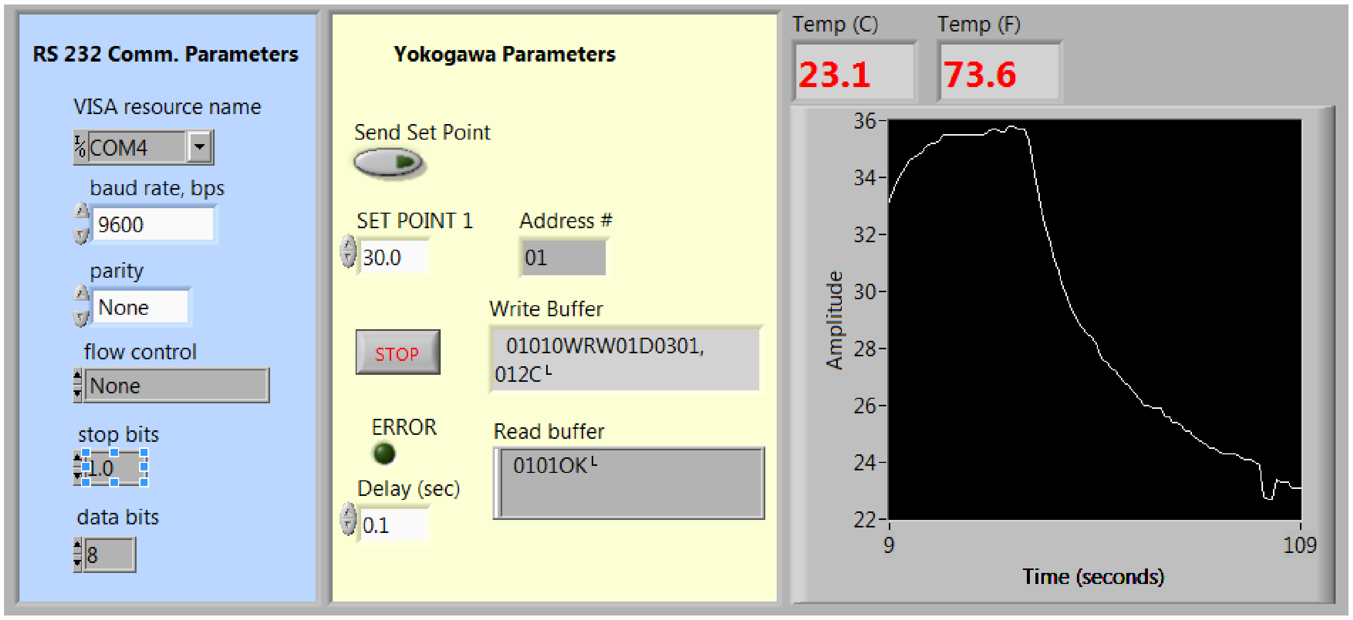

2016
This VI reads the temperature from the Yokogawa UT-350 PID Controller every cycle unless the Send Set Point button is pressed. When the Send Set Point is pressed the temperature set point is changed. The set point is the target output of the PID control algorithm.
1) Initialize
VISA resource name
VISA Configure Serial Port
Baud rate
data bits
parity
stop bits
flow control
Address #
Yokogawa ID # (1 to 31)
Send Set Point
Delay (sec)
True
Hex-ETX 02
Computer # 01
Function 0
Write Mode WRW
# of words 01
Function Address D0301
SET POINT 1
10
Number To Hexl String
4
TRUE State
When Send Set Point is Pressed the Temperature is not Read for One Cycle and the Set Point is changed instead.
Build Message to Set Temperature Set Point
Hex-ETX 03
Carriage Return
Concatenate Strings
VISA Write
2) Write Message
3) Wait
Time Delay2
4) Read ALL Bytes on the Port
Property Node
Instr
Bytes at Port
VISA Read
If OK is in the returned message the Set Point was received and set.
5) Parse Serial Response
Match Pattern
OK
-1
Write Buffer
ERROR
Read buffer
VISA Close
Wait (ms)
1000
stop

Figure 7.16: Snippet of Yokogawa UT-350 Controller program to change the set point.


RS 232 Comm. Parameters

VISA resource name: COM4

baud rate, bps: 9600

parity: None

flow control: None

stop bits: 1.0

data bits: 8

Yokogawa Parameters

Send Set Point

SET POINT 1: 30.0

Address #: 01

Write Buffer: 01010WRW01D0301, 012C

Read buffer: 0101OK

ERROR

Delay (sec): 0.1

Temp (C): 23.1

Temp (F): 73.6

Figure 7.17: Front Panel of Yokogawa UT-350 Controller showing message when set point is changed.

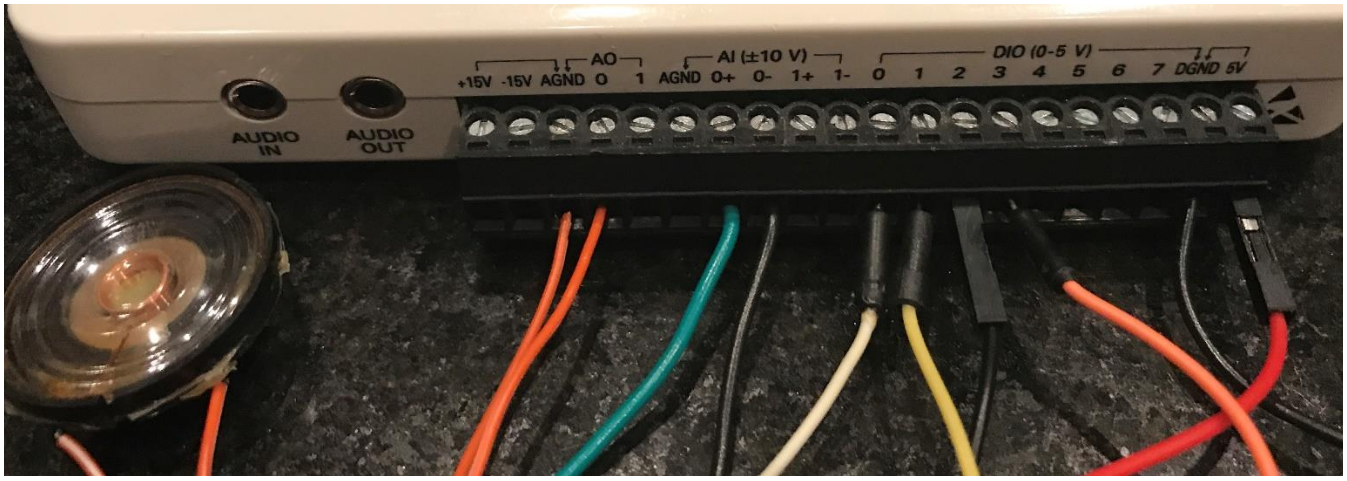
Module 7 – References and Links

The following references and links provide supporting information for this module. All references in this module are either cited within the module inside brackets or URL links are embedded in hyperlinks or fully listed.

[1] Data Communication Overview: https://en.wikipedia.org/wiki/Data_transmission
[2] Communication Protocol Overview: https://en.wikipedia.org/wiki/Communication_protocol
[3] Serial Communication Overview: https://en.wikipedia.org/wiki/Serial_communication
[4] Parallel and Serial Information: https://en.m.wikipedia.org/wiki/Parallel_port
[5] GPIB (IEE-488) Overview: https://en.wikipedia.org/wiki/IEEE-488
[6] TCP/IP Overview: http://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf
[7] USB Overview: https://en.wikipedia.org/wiki/USB
[8] USB Implementers Forum: http://www.usb.org/home
[9] I^2C Overview: https://en.wikipedia.org/wiki/I%C2%B2C
[10] I^2C Tutorial: https://learn.sparkfun.com/tutorials/i2c
[11] Bluetooth Overview: https://en.wikipedia.org/wiki/Bluetooth
[12] Bluetooth Technology Website: https://www.bluetooth.com/
[13] RS-232 Overview: https://en.wikipedia.org/wiki/RS-232
[14] One page overview of the RS-232 specification: https://www.omega.com/techref/pdf/RS-232.pdf
[15] RS-422 Overview: https://en.wikipedia.org/wiki/RS-422
[16] RS-485 Overview: https://en.wikipedia.org/wiki/RS-485
[17] Single Ended versus Double Ended: https://www.allaboutcircuits.com/textbook/semiconductors/chpt-8/single-ended-differential-amplifiers/
[18] Texas Instrument Video that explains CMRR, PSRR, SNR, THD: https://training.ti.com/ti-precision-labs-adcs-ac-dc-specifications?HQS=TI-null-null-eetech-vids-tr-dynamic-wwe
[19] Serial Communication details: https://en.m.wikipedia.org/wiki/Asynchronous_serial_communication
[20] Mathworks, Baud Rate: https://www.mathworks.com/help/matlab/matlab_external/baudrate.html
[21] Mathworks, Overview of the Serial Port: https://www.mathworks.com/help/matlab/matlab_external/overview-of-the-serial-port.html#f84557
[22] IBM knowledgebase articles on parity checking: https://www.ibm.com/support/knowledgecenter/en/ssw_aix_61/com.ibm.aix.networkcomm/asynch_params_parity.htm
[23] Flow Control Overview: https://en.wikipedia.org/wiki/Flow_control_(data)#Hardware_flow_control
[24] RS-232 Overview and RTS/CTS explanation: https://en.wikipedia.org/wiki/RS-232#RTS,_CTS,_and_RTR
[25] Little-Endian and Big-Endian explained: https://en.m.wikipedia.org/wiki/Endianness
[26] Error Detection: https://www.mathworks.com/help/comm/ug/error-detection-and-correction.html
[27] National Instruments, NI-VISA Overview: http://www.ni.com/tutorial/3702/en/
[28] Null Modem Overview: https://en.wikipedia.org/wiki/Null_modem
[29] Fluke 8845A/8846A Programming Manual: http://assets.fluke.com/manuals/8845a___pmeng0200.pdf
[30] Omega Thermocouple Color Codes: https://www.omega.com/techref/colorcodes.html
[31] Yokogawa UT-350 Programming Manual: https://web-material3.yokogawa.com/IM05D01D21-10E_002.pdf
[32] Comparison between asynchronous and synchronous communication: https://en.m.wikipedia.org/wiki/Comparison_of_synchronous_and_asynchronous_signalling
[33] Parallel Port: https://en.wikipedia.org/wiki/Parallel_port
[34] Ethernet Overview: https://en.wikipedia.org/wiki/Ethernet
[35] RS-232 versus TTL Logic Levels: https://www.sparkfun.com/tutorials/215
[36] Parity Checking Overview: https://en.wikipedia.org/wiki/Parity_bit
[37] UART Overview: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

Appendix A – Step by Step Guide to Digital, Analog, and Counter IOs in a DAQ

This section will go through a step by step process of how to use DAQ analog, digital, and counter inputs and outputs with a myDaq. The process will be nearly the same for all other NI DAQs. The first step is to connect wires to the pins in the myDaq so they can be connected to the circuit to take measurements. The following image shows the connections. The myDaq 5V (red wire) is used to power the circuit and will be connected to the + rail on the breadboard. The two plain black wires (DGND and AO-) are connected to ground on the breadboard. **Note:** *All 8 D**IO** pins can be configured as either digital **I**nputs or **O**utputs, but some of these lines are also used for counters.* For example, any of the 8 DIO lines can be used as a **counter input**. In Example 1 the rising edge pulses are counted with DIO-0 in order to estimate the rpm. For **counter outputs**, DIO-3 is the only pin that can be used. In Example 2, an LED is flashed by connecting the counter output (DIO-3) to the LED.



Connection Descriptions and how they are used – See this link for the myDaq pin out.

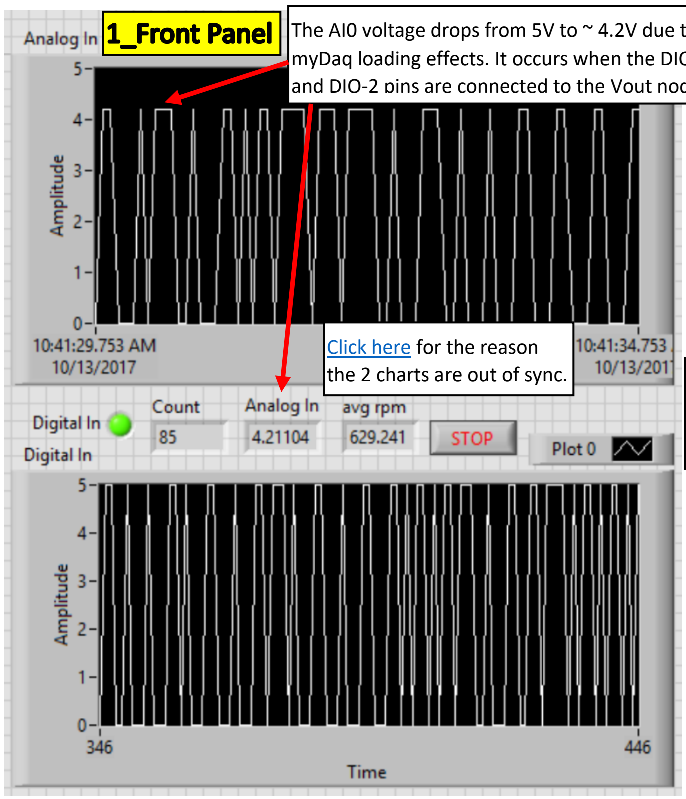
1. **Digital Inputs** – DIO-line 2 (black wire with plastic connector). Used in Ex. 1 to measure the digital Vout.
2. **Analog Input** – AI0+ (green wire) and AI0- (plain black wire). Used in Ex. 1 to measure the analog Vout.
3. **Counter Input** – DIO-line 0 (white wire). The counter input pin is used in Ex. 1 to count the # of pulses.
4. **Digital Output** – DIO-line 1 (yellow wire), DIO-line 3 (orange wire). In Ex. 1, DIO-1 is used to turn on a green LED when the analog voltage is greater than 2V (TTL High Logic Level) and DIO-line 3 is used to turn on a red LED when the digital voltage level (using TTL levels) is high. In Ex. 2, DIO-3 is used to flash a larger green LED.
5. **Analog Output** – A speaker is controlled in Ex. 2 by connecting it to analog output 0 (ao0) & ground (AGND).
6. **Counter Output** – DIO-line 3 (orange wire). As previously noted, this line is used in Ex. 2 to flash an LED.

Appendix A - Example 1) Digital, Analog, and Counter **Inputs** (a Digital Output with 2-channels is also shown)

Appendix A Example 1 will show the following 4 steps. After this example, an additional step by step guide (Appendix A Example 2) is provided to cover all of the output types (digital, analog, and counter).

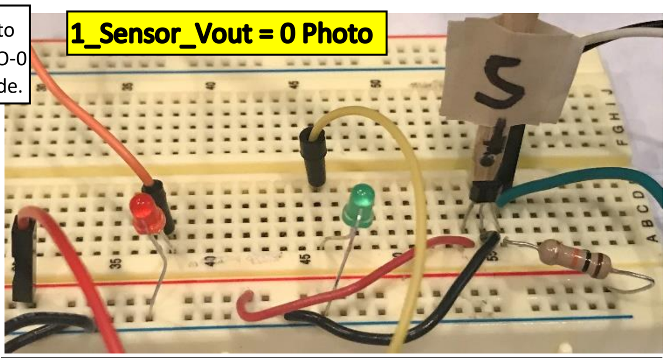
1. How to measure the digital voltage level from the Hall Effect sensor with the **digital Input** pin.
2. How to measure the exact voltage from the Hall Effect sensor with the differential **analog Input** pins.
3. How to count the number of times the voltage changes from low to high using the **counter Input** pin.
4. How to turn on two LEDs using **digital output** pins when the voltage in parts 1 and 2 is a TTL high level.

First, the final block diagram & front panel are shown as a magnet is rapidly spun in front of a Hall Effect sensor.

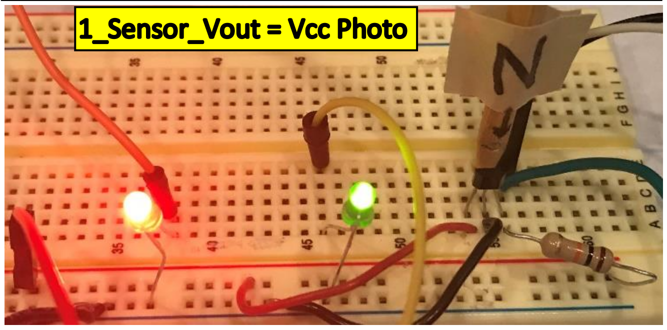


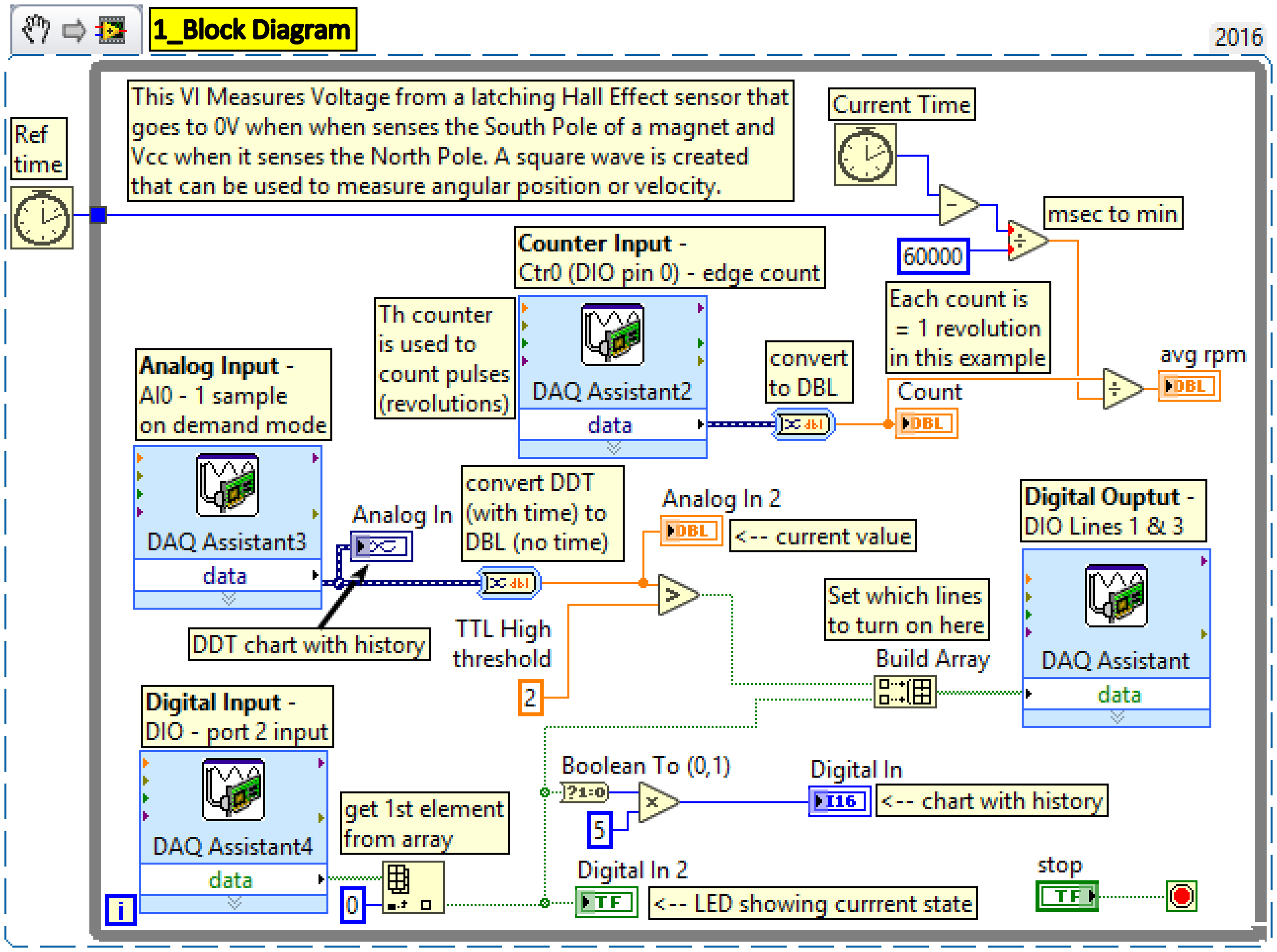
The AI0 voltage drops from 5V to ~ 4.2V due to myDaq loading effects. It occurs when the DIO-0 and DIO-2 pins are connected to the Vout node.

Click here for the reason the 2 charts are out of sync.



The photo above shows the Hall Effect sensor outputting 0V (LEDs off) when detecting the south pole of a magnet. The photo below shows 5V is output (LEDs On) when the North pole is detected.




1_Block Diagram
2016
Ref time
Current Time
msec to min
60000
Counter Input - Ctr0 (DIO pin 0) - edge count
Th counter is used to count pulses (revolutions)
DAQ Assistant2
data
convert to DBL
Each count is = 1 revolution in this example
Count
avg rpm
Analog Input - AI0 - 1 sample on demand mode
DAQ Assistant3
data
Analog In
convert DDT (with time) to DBL (no time)
Analog In 2
<-- current value
DDT chart with history
TTL High threshold
2
Set which lines to turn on here
Build Array
Digital Ouptut - DIO Lines 1 & 3
DAQ Assistant
data
Digital Input - DIO - port 2 input
DAQ Assistant4
data
get 1st element from array
Boolean To (0,1)
5
Digital In
<-- chart with history
Digital In 2
<-- LED showing currrent state
stop
0
i

This VI Measures Voltage from a latching Hall Effect sensor that goes to 0V when when senses the South Pole of a magnet and Vcc when it senses the North Pole. A square wave is created that can be used to measure angular position or velocity.

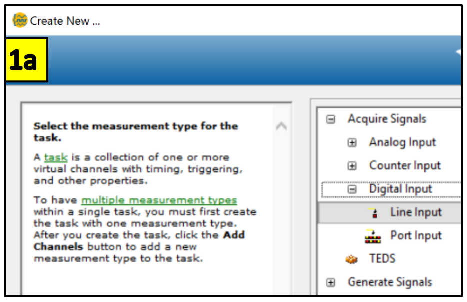
The steps to create the Express VI DAQ blocks in Example 1 are shown below (Steps 1 through 4).

1) **Digital Input** - Since the SS400 series latching Hall Effect sensor outputs 0 to 5V in accordance with TTL logic levels, the output can be directly connected to the DIO pin of the myDaq. As the north pole of the magnet is detected by the sensor, the DIO pin (DIO-2, black wire with plastic connector, is used in this example) will change to a Boolean True (5V) and when it detects the south pole it will change to a Boolean False (0V).

a) After clicking on the DAQ Assistant, select a digital line input (see **1a** below) to turn on one or more digital lines. **Note:** *the "port input" includes all 8 DIO pins on the myDaq.*

b) Next, select the desired line (see **1b**). For this example, DIO-line 2 is selected as showed previously.

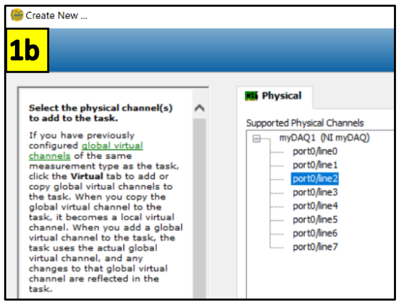
c) After clicking on OK, the **1c** GUI will appear. In this example, the default values are all that is needed.



Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

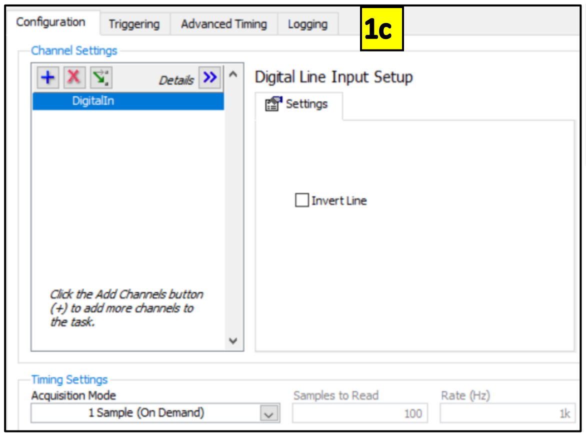
To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.



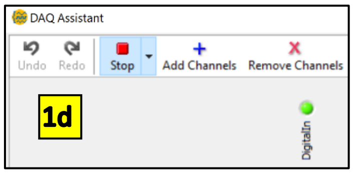
Select the physical channel(s) to add to the task.

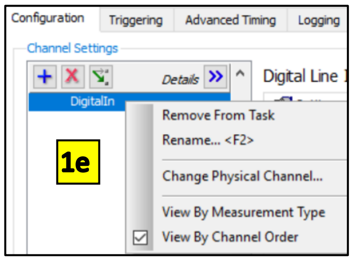
If you have previously configured global virtual channels of the same measurement type as the task, click the **Virtual** tab to add or copy global virtual channels to the task. When you copy the global virtual channel to the task, it becomes a local virtual channel. When you add a global virtual channel to the task, the task uses the actual global virtual channel, and any changes to that global virtual channel are reflected in the task.

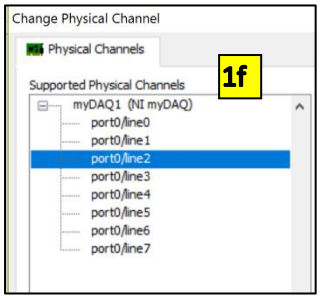
1d is actually above **1c** when the DAQ assistance is created and it gives the ability to add or remove channels and test your selected channels by clicking the Run button (It changes from "Run" to "Stop" after clicking on it). **1d** shows the LED is On for "DigitalIn" (default channel name) This means the measurement is > 2V.

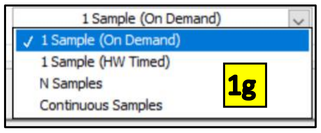


Click the Add Channels button (+) to add more channels to the task.









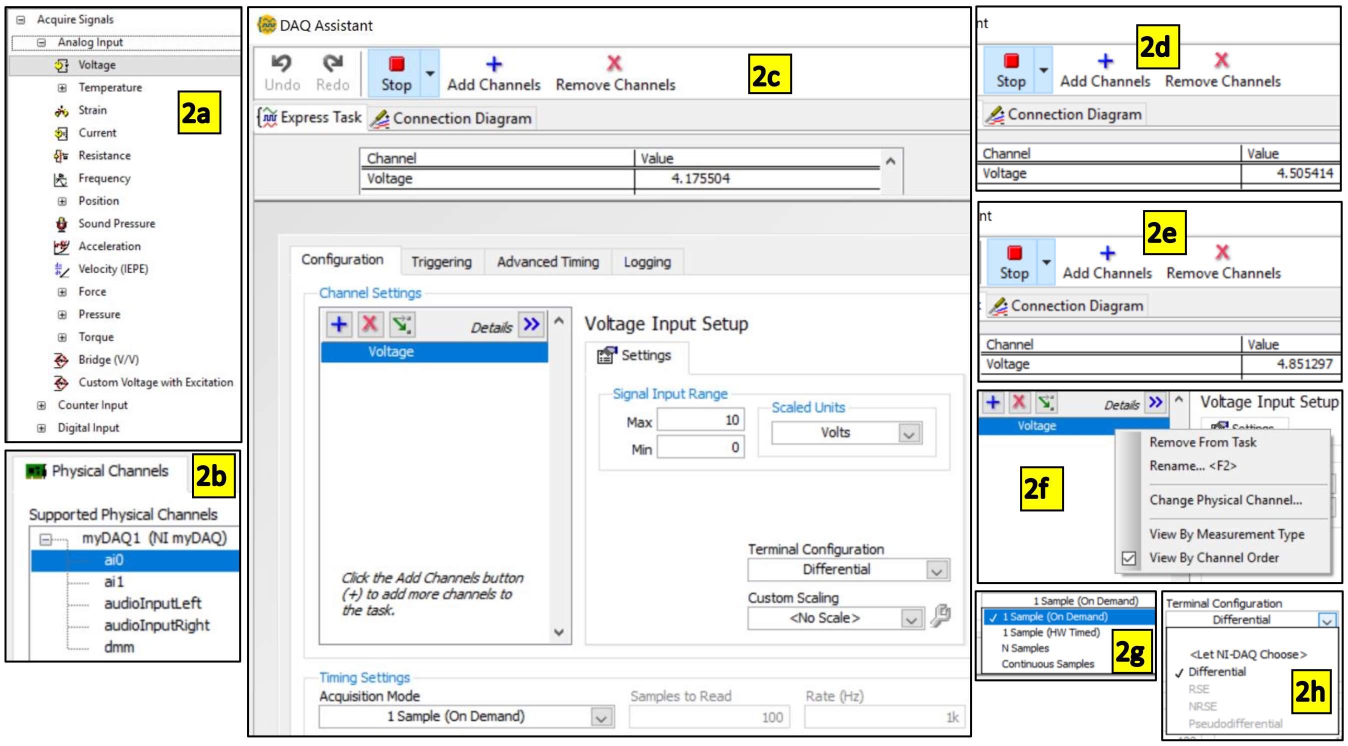
- If you end up needing to change the channel number or rename the channel, you can right click on the "DigitalIn" name in **1c** and the **1e** GUI will appear with 5 options.
- If you select "Change Physical Channel," GUI **1f** appears to allow which channel to select (port0/line 2 is currently selected so it is shaded blue).
- If you want to always flip your logic on that line then you can select the "Invert Line" check box in **1c**.
- And lastly, the default **Acquisition Mode** in **1c** is "1 sample (On Demand)". This is the easiest method to use. It will read one sample into the DAQ as either a true or a false Boolean value each time the DAQ assistant VI is called in the program. There are 3 other options, as shown in **1g**. "N Samples" is also a commonly used in acquisition mode. An example of how the N samples option works is shown in the next bullet.
- If you wanted to acquire a block of 500 digital samples then you could set the number of samples to 500 and use N sample mode. The "Rate" must be selected so the DAQ will know how fast to acquire the 500 samples. If 200 Hz was selected then every 5 ms (1/200) a sample will be acquired and all 500 samples will be acquired in 2.5 seconds (500 samples * 5 ms). After the 500 samples are acquired, no more data will be acquired until the block is called again. If the DAQ assistant is in a while loop then it will be continuously called each iteration and acquire 500 samples each time it is called. There will be time gaps between each time the block is called while the other operations in that while loop iteration are completed.

Table of Contents

2) **Analog Input** – To measure the actual voltage of the Vout pin of the SS400 series latching Hall Effect sensor, an analog measurement is used. AI0+ (green wire) is connected to the Vout pin of the sensor as seen in the previous photos. The AIO- (plain black wire) is connected to ground because the Vout voltage is referenced to ground. The myDaq only allows differential measurements (AI+ and AI-) for both of its analog input channels. In this case, the AI0- pin is connected to ground, but having ground as a reference is not a requirement with differential measurements. For example, the voltage across a resistor (that has neither leg connected to ground) could be measured by connecting the AI+ and AI- pins to the two sides of the resistor.

a) GUI **2c** will appear after selecting analog input, voltage, and ai0 in the DAQ assistant step by step wizard, shown in **2a** and **2b**. Notice in **2a**, voltage is one of 15 different analog input options. The DAQ wizard has built-in measurement wizards to help set up the DAQ for different types of sensors.

b) The only change that was made to the default in this example was to the "Signal Input Range". The default is the maximum and minimum range of the myDaq, which is ± 10V. Since the Hall Effect sensor will never have a negative voltage, -10 was changed to 0 in **2c**. The maximum limit could have been reduced to 5V, but to be safe it was left at 10V, because it would still be a valid range if the power source was changed to an external 9V battery. By reducing the range, the resolution is improved as shown in Equation 3.2.



2c: Channel: Voltage, Value: 4.175504. Signal Input Range: Max 10, Min 0. Scaled Units: Volts. Terminal Configuration: Differential. Custom Scaling: <No Scale>. Acquisition Mode: 1 Sample (On Demand). Samples to Read: 100. Rate (Hz): 1k.

2d: Channel: Voltage, Value: 4.505414.

2e: Channel: Voltage, Value: 4.851297.

- If you need to change the channel number or rename the channel, you can right click on "voltage," which is the default channel name, and the **2d** GUI will appear. If the "Change Physical Channel" option is selected in **2f**, the **2b** GUI will appear showing that ai0 is currently selected. The ai1 (analog input 1 channel) could also be selected, but the audio inputs left/right and the dmm (Digital Multimeter) would not work the same way as the differential analog inputs (ao0 and ao1).
- The N Sample Acquisition Mode is discussed for digital inputs on the last bullet of the previous page. That information also applies to analog inputs.
- When you click on the "Terminal Configuration" drop down box in **2c,** the GUI in **2h** appears, which shows that the only possible choice for the myDaq is "Differential" and the other options are grayed out so they can't be selected. Even though the myDaq has the ability to take differential measurements, in this example the negative analog input (AI0-) is grounded, so it is not much different than the single ended measurement

options, referenced single ended (RSE) or non-referenced single ended (NRSE). The NRSE can have a reference that is floating (not tied to ground) and uses the AI_sense line as its reference. The following white paper goes into more depth on this topic: http://www.ni.com/white-paper/3344/en/

- Another important note is that the "Run/Stop" button allows the analog voltage to be measured instead of only the Boolean T/F input (as shown in the Figure labeled **1d** in the digital input section on the previous page). When the analog input pin is the only pin connected to Vout of the Hall Effect, the voltage was over 4.85 V (**2e**), which was nearly the exact voltage level of the power rail. When the counter input pin from DIO-0 was added in the same node on the breadboard as AI0+, the level dropped to ~ 4.5V due to loading effects, as seen in **2d**. When the digital input line (DIO-2) was also added the voltage level drops even further (4.1755 V) as shown at the top of **2c**. Loading effects is an important issues with DAQ systems.

Note: *If the second analog input (ai1), that is available in the myDaq, was added by clicking on the blue plus symbol, both voltage readings would show up at the top of the screen in the test area.*

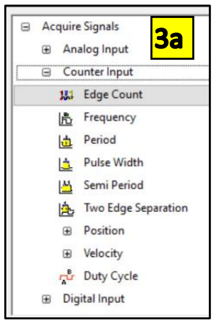
- The analog input goes high and low at approximately the same time as the digital input. However, the charts in the final block diagram shows that the two inputs are **out of sync**. This is due to the data type used for the analog chart being set to DDT (dynamic data type) and the digital chart being set to DBL (double precision float). DDT data type includes timing information, so the analog input chart's x axis is equal to time, while the digital chart only has the sample number for the x-axis. The time per sample is not the same between these two charts so different amounts of samples are displayed on them.

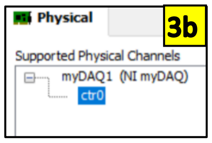
3) **Counter Input** - As the wooden stick with a magnet inserted into it (shown in this photo) is rotated, Vout from the sensor will be used to change from high to low every revolution. The number of pulses is measured by configuring DIO-0 as an edge counter input. Once the number of pulses is known the rpm is estimated by dividing the number of revolutions (which is equal to the number of high pulses in this experiment) by the number of minutes that has elapsed.

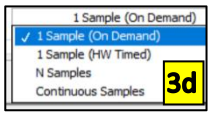
a) When configuring the DAQ assistant, **acquire-counter-edge count** is selected (as shown in **3A**).

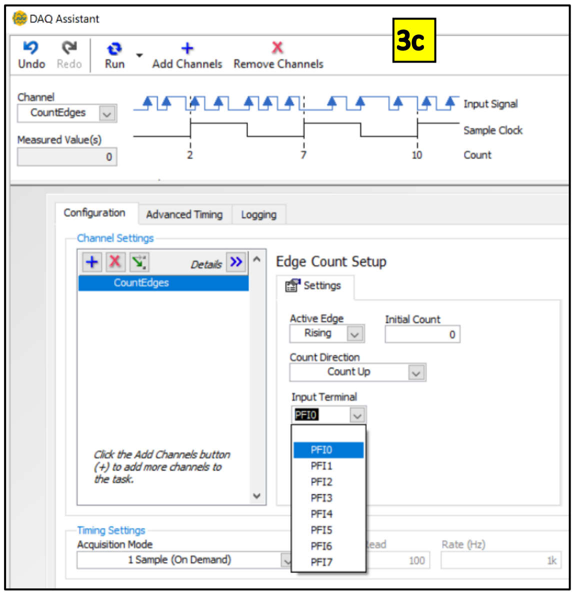
b) The myDaq only has one counter, so ctr0 must be selected (as shown in **3b**).

c) Once selecting ctr0 the default settings shown in **3c** appears. The default settings are used in this example.









Channel: CountEdges
Measured Value(s): 0
Active Edge: Rising
Initial Count: 0
Count Direction: Count Up
Input Terminal: PFI0
Acquisition Mode: 1 Sample (On Demand)
Samples to Read: 100
Rate (Hz): 1k

- All the previously described Aquistion Mode info (see **3d**) applies also to counters.

- When testing the counter by pressing the run button the Measured Value will increase by 1 each time a new **rising edge** is detected.
- The "Active Edge" setting allows for the selection of rising edge or falling edge.
- The "Initial Count" is 0 by default, but can be initalized at a different value, if desired.
- Even though there is only one counter, any of the 8 pins shown in the drop down box in **3c** can be used. PFI stands for Programmable Function Interface and PFI 0 to 7 are tied to the physical channels DIO 0 to 7. The counter input can be used on any of the 8 DIO pins.

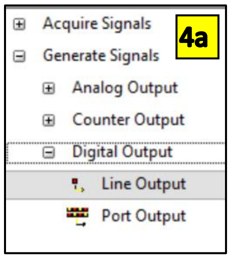
4) **Digital Output (2 channels)** – In addition to the three types of DAQ inputs, this example also has digital outputs. They were added to this example (focused on DAQ inputs) to allow LEDs to be turned on to visually show on the breadboard when the north pole of the magnet came in contact with the Hall Effect sensor. The red and green LEDs (see photos here) turn on when a TTL logic high is detected by the analog input (AI0+ & AI0-) and the digital input (DIO-2), respectively. The trickiest part of this programming is the Boolean array data type (thick green line) that is used in LabVIEW for digital inputs and outputs. In the block diagram when DIO-2 was read from the sensor it came out of the DAQ assistant as an array that contained only one Boolean variable. If multiple DIO pins were read in from the DAQ then the array would contain multiple Booleans. An "Index Array" block can be added, and the element of the array set to 0 so that only the first element is acquired. In contrast, the reverse process is followed when writing to a digital output pin. All inputs are wired into the "build array" block in the order that you want them to be controlled (see Block Diagram). To add more wires to the build array block, you can resize it by dragging down on it in the middle.

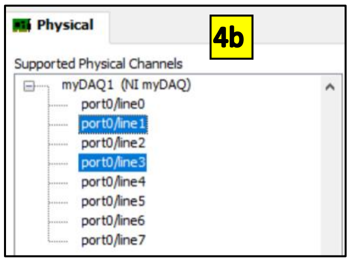
a) When configuring the DAQ assistant, select **Generate-Digital Output-Line Output** (as shown in **4A**).

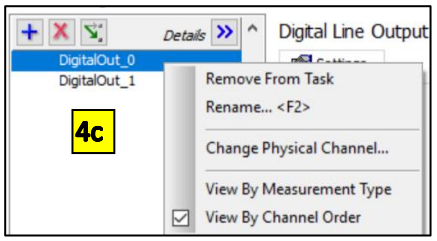
b) Press and hold the control key and click on lines 1 and 3 in **4b**, because the LEDs are physically wired to those 2 lines (See Photo).

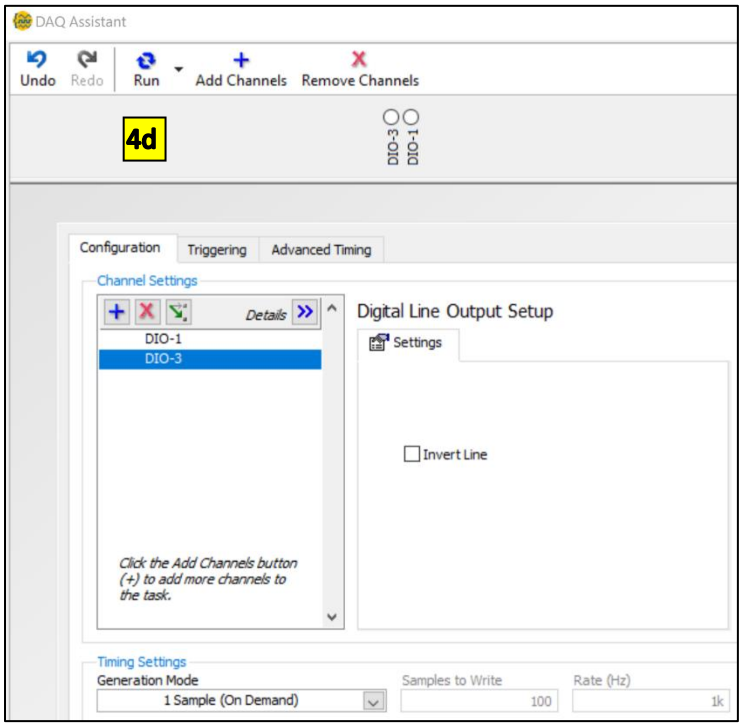
c) Since there are two channels, it is a good idea to rename the lines to the DIO numbers 1 and 3 by right clicking on each channel in **4c** and selecting "Rename." They are renamed to DIO-1 and DIO-3 in **4d**. You can also add more channels with either of the blue plus symbols and remove channels with either of the red Xs.

d) The final GUI is shown in **4d**. If the Run button is pressed one or both of the DIO pins can be turned on (**4e**).

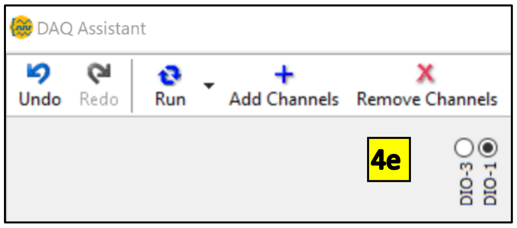








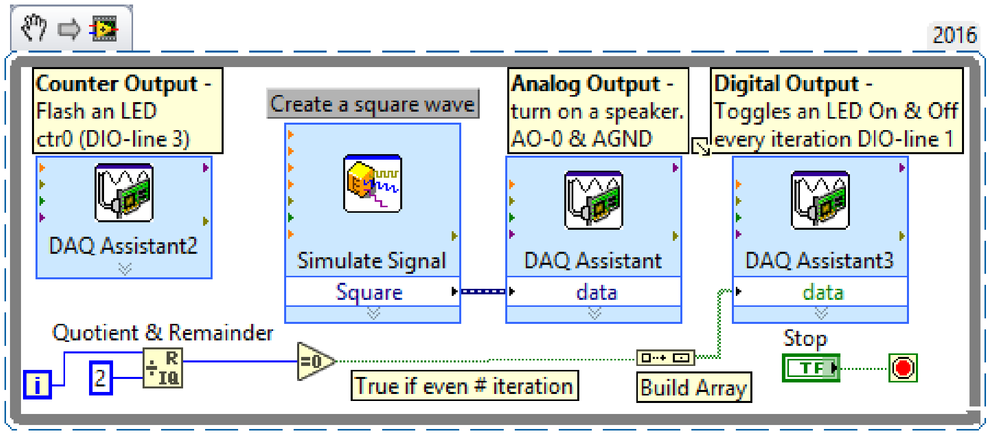
Generation Mode: 1 Sample (On Demand)
Samples to Write: 100
Rate (Hz): 1k

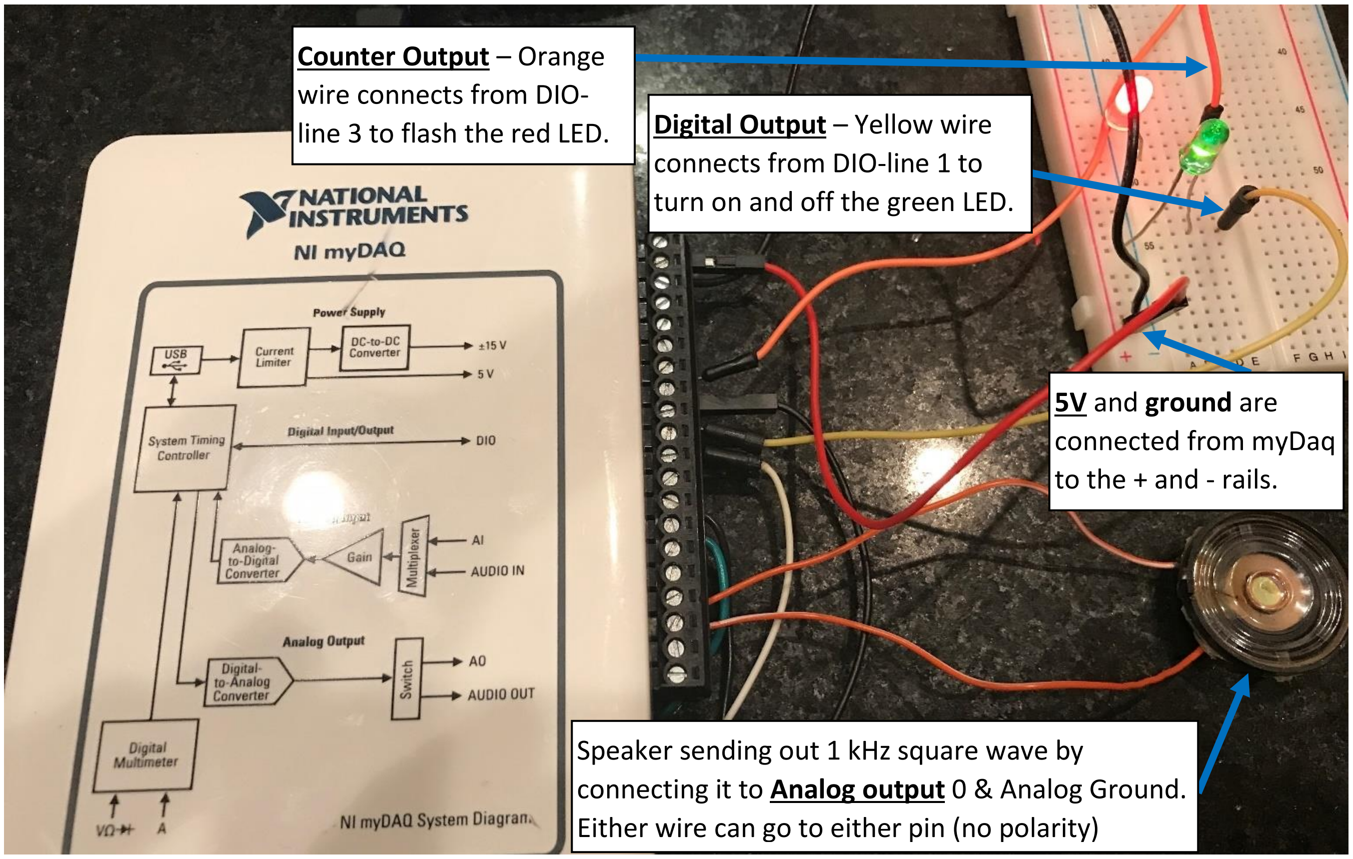


- **Generation Mode** for the digital, analog and counter outputs has the same options and functionality as the **Aquistion Mode** for digital, analog and counter inputs (see digital input section for details).
- As described for digital inputs, the Invert line check box (shown in **4d**) can be used to invert (or flip) the logic on one or more channels. For example, if DIO-1 has its "Invert Line" box checked, but DIO-3 has its unchecked and a true is sent into the DAQ assistant VI to both channels then the DIO-1 pin would be set to 0V and the DIO-3 pin would be set to 5V.

Appendix A - Example 2) Digital, Analog, and Counter Outputs

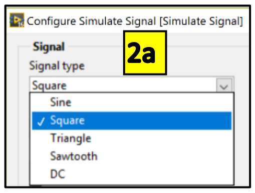
The front panel will not be shown for this example because these DAQ VIs are outputs and the only item on the front panel is a stop button. In the pages following the photo below, the four express VIs that are used in the program are explained. The wiring diagram and high-level description of each operation is described at this bookmark and also explained in textboxes in the photo below.

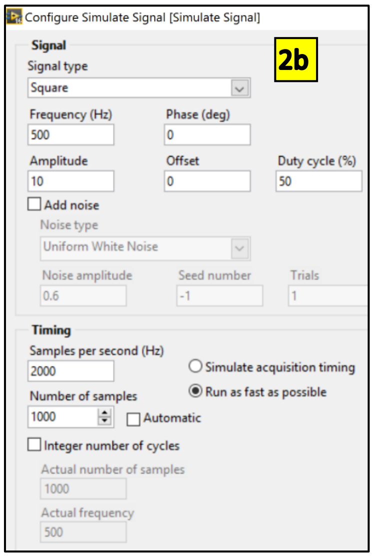

2016
Counter Output - Flash an LED ctr0 (DIO-line 3)
DAQ Assistant2
Create a square wave
Simulate Signal
Square
Analog Output - turn on a speaker. AO-0 & AGND
DAQ Assistant
data
Digital Output - Toggles an LED On & Off every iteration DIO-line 1
DAQ Assistant3
data
Quotient & Remainder
True if even # iteration
Build Array
Stop


Counter Output – Orange wire connects from DIO-line 3 to flash the red LED.
Digital Output – Yellow wire connects from DIO-line 1 to turn on and off the green LED.
5V and **ground** are connected from myDaq to the + and - rails.
Speaker sending out 1 kHz square wave by connecting it to **Analog output** 0 & Analog Ground. Either wire can go to either pin (no polarity)
NATIONAL INSTRUMENTS
NI myDAQ
NI myDAQ System Diagram

1) **Digital Output (1 Channel)** – Example 1 had a digital output with 2 channels so that discussion will be referred to instead of re-writing it. When only one digital output is needed only one Boolean is wired into the build array block. In this example, the Boolean value is set to alternate each iteration by dividing the while loop iteration (i) counter by two and taking the remainder of that result and comparing it to zero.

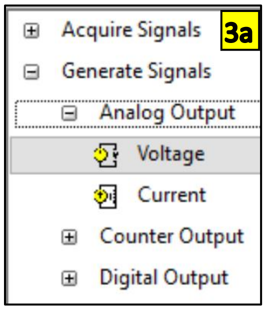
2) **Simulate Signal Express VI** – This VI can be used to create a sine wave, square wave, triangle, sawtooth, or DC signals as shown in **2a**. The specified **Signal** properties (i.e. amplitude, frequency, noise, etc.) can be adjusted and the **Timing** options can be used to set a specific number of points and sampling rate (or how closely in time the points are spaced) as shown in **2b**. The Simulate Signal GUI also includes a signal preview function (not shown here) that allows the created signal to be viewed in real time as the **Signal** or **Timing** vales are adjusted.

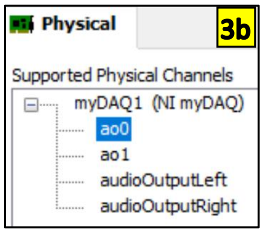


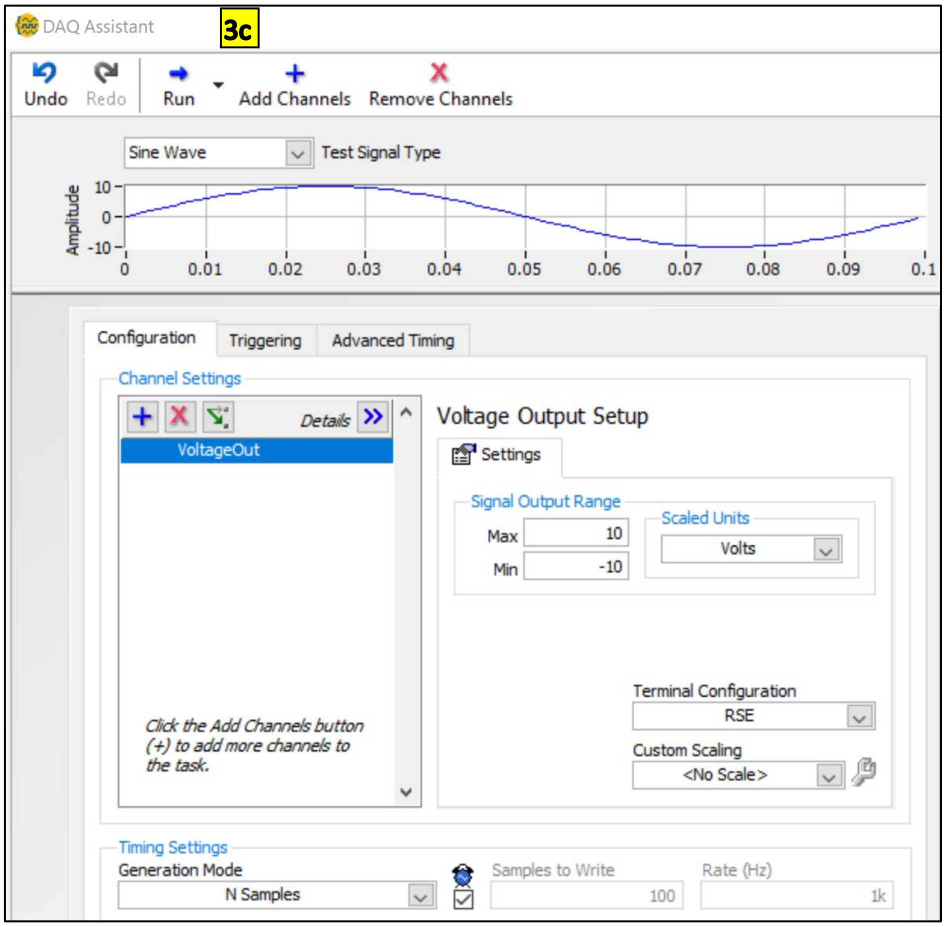


Signal
Signal type: Square
Frequency (Hz): 500
Phase (deg): 0
Amplitude: 10
Offset: 0
Duty cycle (%): 50
☐ Add noise
Noise type: Uniform White Noise
Noise amplitude: 0.6
Seed number: -1
Trials: 1
Timing
Samples per second (Hz): 2000
○ Simulate acquisition timing
● Run as fast as possible
Number of samples: 1000
☐ Automatic
☐ Integer number of cycles
Actual number of samples: 1000
Actual frequency: 500

3) **Analog Output** – When configuring the DAQ assistant for analog output, select **Generate-Analog Output-Voltage** (as shown in **3a**). Then, select the channel (see **3b**). In this example ao0 was selected, but ao1 could have been selected instead since it is the same type of output. There is also an audio cable jack on the myDaq that can be used to connect devices with 2-channel stereo sound (or mono if only the audioOutputLeft is used). The blue clock is checked in **3c**, which means the **Timing** is set by the incoming signal. This was set in step 2).



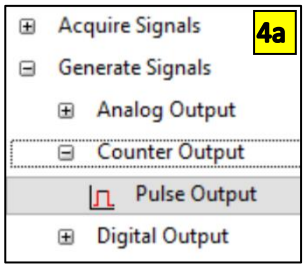


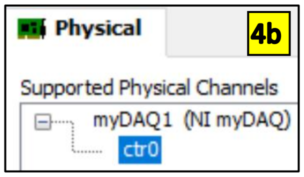


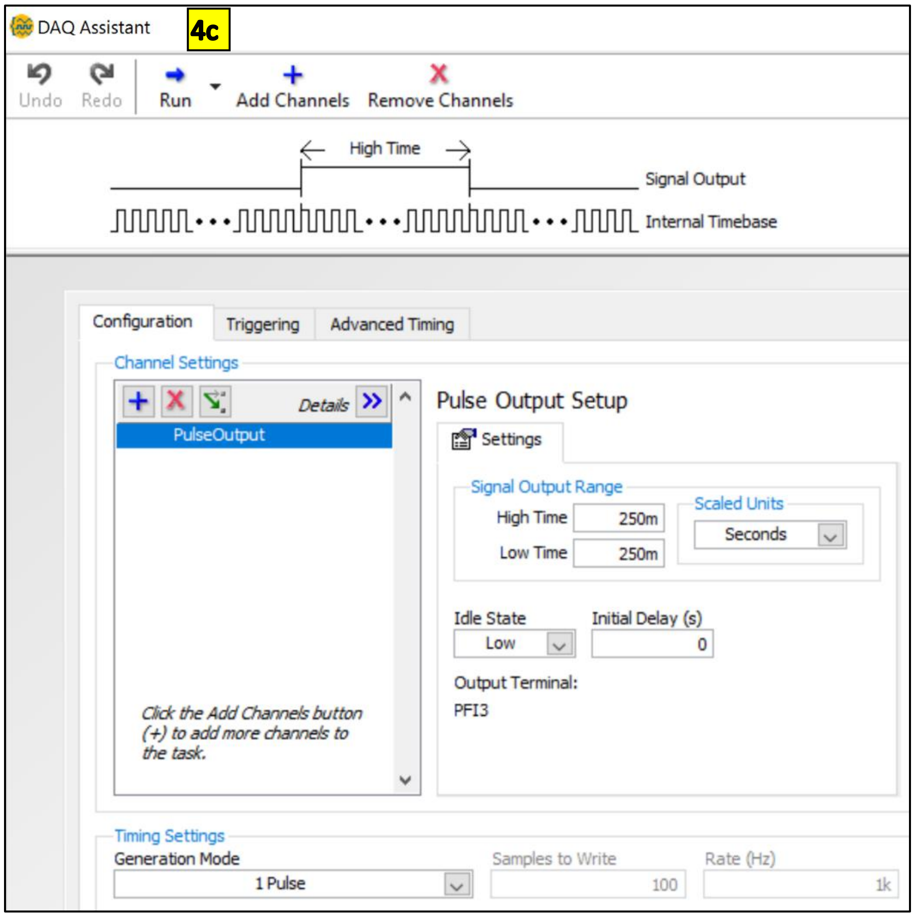
Signal Output Range
Max: 10
Min: -10
Scaled Units: Volts
Terminal Configuration: RSE
Custom Scaling: <No Scale>
Timing Settings
Generation Mode: N Samples
Samples to Write: 100
Rate (Hz): 1k

4) __Counter Output__ – When configuring the DAQ assistant for a counter output, select **Generate-Counter Output-Pulse Output** (as shown in **4A**). Then, ctr0 must be selected (see **4b**) because the myDaq only has one counter.

- Right clicking on the channel name allows you to rename it from the default name of "PulseOutput."
- The high time and low time can be set so that each pulse is sent out according to desired specifications. In this example, "1 Pulse" is selected so it will only flash the LED one time each time the DAQ assistant VI is called (i.e. each iteration of the while loop). If "N Pulses" is selected, then an LED can be flashed multiple times.







Click the Add Channels button (+) to add more channels to the task.

High Time 250m
Low Time 250m
Scaled Units: Seconds
Idle State: Low
Initial Delay (s): 0
Output Terminal: PFI3
Generation Mode: 1 Pulse
Samples to Write: 100
Rate (Hz): 1k

- Additional DIO pins can be utilized for more complicated counter operations, such as a quadrature encoder. The following link shows that the myDaq DIO pins 0 to 4 are specified to act as the counter SOURCE, GATE, AUX, OUT, and FREQOUT functions in that order. http://www.ni.com/documentation/en/mydaq-student-data-acquisition-device/latest/mydaq/pinout/
- The following tutorial shows how to configure a quadrature encoder in a DAQ. https://www.ni.com/getting-started/set-up-hardware/data-acquisition/quadrature-encoders